



Length-Aware Regular Expression Solving in Z3str3

Murphy Berzish, Vijay Ganesh
University of Waterloo



Outline

- Motivation for Length-Aware Techniques
- Theory Signature
- Theoretical Results
- Algorithm and Length-Aware Heuristics
- Experimental Evaluation
- Conclusions and Future Work



Motivation

- Many applications (e.g. user input sanitizers) require string logics supporting regex terms and integer constraints
- Theory of regex constraints with string length, arithmetic, concatenation, and string-number conversion is undecidable
- Regex constraints typically handled by reduction to automata or to word equations
 - Automata-based approaches powerful but expensive
 - Many regular languages cannot be expressed using word equations alone
- Key insight: formulas obtained from practical applications have both **implicit** and **explicit** length information -- use this to make automata-based representation efficient



Challenges of Automata-Based Techniques

- Automaton representation is very natural for regular expression constraints
- Two key practical challenges:
 - Length constraints from word equations are conjunctions; length constraints from regular expressions are disjunctions
 - Automata operations such as intersection are very expensive, but they are required
- Address these challenges through **lazy extraction of implied length constraints**
- Use length information to prune the search space and avoid expensive operations
- Similar techniques for word equation solving have been extremely effective (cf. Z3str2)



Theory Signature

- Theory $T_{\text{LRE},n,c}$
- Quantifier-free many-sorted first-order theory of:
 - linear integer arithmetic over string length function (L)
 - regular expression membership predicates (RE)
 - string-number conversion (n)
 - string concatenation (c)
- And its fragments:
 - $T_{\text{LRE},n}$ has everything except concatenation
 - $T_{\text{LRE},c}$ has everything except string-number conversion
 - T_{LRE} has no concatenation or string-number conversion
- No equalities between string terms, no general word equations



Semantics of String-Number Conversion

- Needed to reason about library functions such as `strtol()`, `Integer.toString()`, etc.
- Expressed with a general predicate `numstr(N, S) : Int x Str -> Bool`
- `numstr(N, S)` is true for a given integer `N` and string `S` iff `S` is a valid binary representation of the number `N`, possibly with leading zeroes
 - `S` only contains the characters `0` and `1`
 - Treating `S` as a place-value representation of binary digits yields a number equal to `N`
- Differences with implementation for standards compliance:
 - Binary used in theoretical presentation, implemented version uses decimal
 - Single predicate vs. `str.to-int` and `str.from-int`



Theoretical Results

The satisfiability problem for the theory $T_{\text{LRE},n}$ is decidable.

Proof sketch: We convert arithmetic (in)equalities to a multi-track automaton with one track per variable appearing in the formula. For each term $\text{numstr}(n, s)$, if n is a numeric constant, then add the regular expression constraint $(s \text{ in } 0^*N)$ where N is the binary representation of n . If n is a variable, then add the regular expression constraint $(s \text{ in } 0^* (0|1) (0|1)^*)$ and substitute s for n on the appropriate track in the automaton. Then convert each regular expression constraint to an automaton, intersect them, and combine with the automaton from the previous step. The original formula is unsatisfiable iff the language accepted by this automaton is empty.



Theoretical Results


The satisfiability problem for the theory $T_{LRE,n,c}$ is undecidable.

Reduce from the theory of power arithmetic, which is quantifier-free linear integer arithmetic extended with the power predicate ($z = x * 2^y$). This theory is undecidable (Büchi 1990) as the power predicate can be used to express arbitrary multiplication (Peano arithmetic).

Reduce each arithmetic term in this theory to its equivalent in $T_{LRE,n,c}$. To encode ($z = x * 2^y$), recall that in binary arithmetic, an unsigned left shift corresponds to multiplication by a power of 2.

Then ($z = x * 2^y$) reduces to ($Y \text{ in } 0^*$) and $\text{len}(Y) = y$ and $\text{numstr}(z, X * Y)$ and $\text{numstr}(x, X)$.

To multiply by a power of 2, we convert a number to its string representation, concatenate a string of zeroes of the appropriate length on the right, and convert back to a number.



Length-Aware Regular Expression Algorithm

- Built on top of a standard DPLL(T) procedure
- Two main phases, which repeat until a solution is found:
 - Check assigned regex constraints to determine satisfiability (and learn length information)
 - Check whether to perform intersections to determine unsatisfiability
- Operations are performed over nondeterministic automata
- Expensive steps (complement, intersection) are performed lazily
 - However, all operations are performed eventually, for completeness
- This algorithm is sound, complete, and terminating



Length-Aware Regular Expression Algorithm

Input: formula of T_{LRE} constraints; Output: SAT or UNSAT

Loop until a solution is found:

- Perform DPLL(T) theory propagation and conflict analysis
- If top-level conflict, return UNSAT
- For each assigned regex membership terms of the form “S in R”:
 - If we have an automaton A for R:
 - If exact length of S is known, enumerate paths of A and discharge to character constraints
 - Refine lower and upper bounds of $\text{len}(S)$ based on A
 - else if $\text{EstimateDifficulty}(R) < \text{threshold}$ or $\text{FailedAttempts}(R) > \text{cutoff}$:
 - Construct automaton A for R
 - else: $\text{FailedAttempts}(R) += 1$
- For each string variable S appearing in any regex constraint:
 - Let A^* be the automaton accepting all strings
 - For each automaton A in a regex constraint over S:
 - If $\text{IntersectionDifficulty}(A^*, A) < \text{threshold}$ or $\text{FailedIntersections}(A) > \text{cutoff}$:
 - $A^* := \text{Intersect}(A^*, A)$
 - else: $\text{FailedIntersections}(A) += 1$
 - If A^* is empty then assert conflict clause
- If length and character constraints satisfied then return SAT



Length-Aware Heuristics

Lower and Upper Bound Refinement

- Use the structure of automata to refine lower/upper bounds on the length of a solution
- Perform breadth-first search to find accepting paths longer than a lower bound or shorter than an upper bound
- We can refine the bound or possibly generate a conflict clause



Length-Aware Heuristics

Difficulty Estimation

- NFA intersection and complement are expensive
- Perform these operations lazily if possible
- For completeness, all operations must be performed eventually
- Estimate the “difficulty” by approximating the final number of states without constructing
- Advantage: performing simpler operations earlier gives us more information



Experimental Results

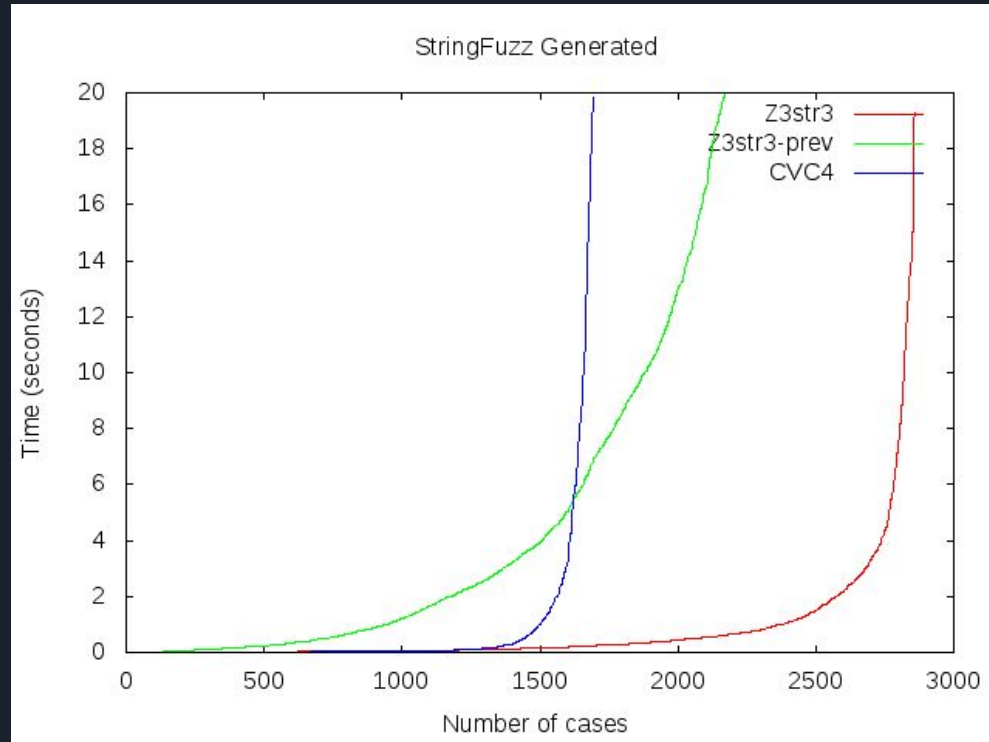
- We compared against CVC4 1.6 and the previous stable version of Z3str3
- Timeout set at 20 seconds per instance
- Models cross-verified between solvers



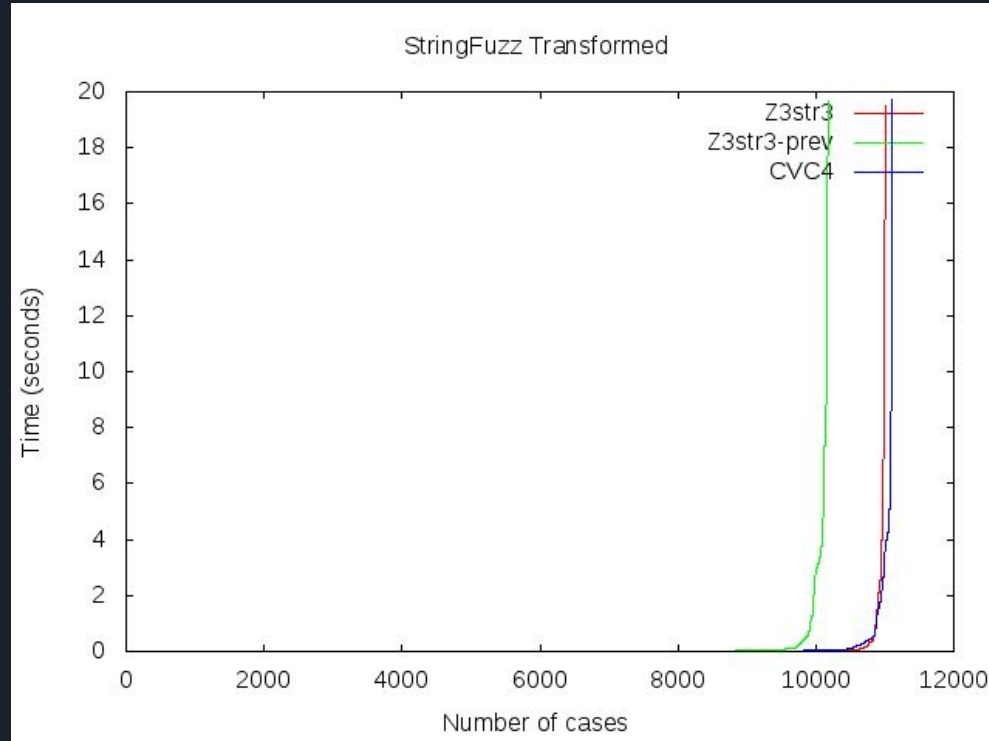
Benchmarks Used

- Evaluated on two suites of regex benchmarks generated by the StringFuzz tool
- `stringfuzz-regex-generated`
 - 3450 randomly generated problems
 - Includes random regular expressions, intersecting constraints, negated regex constraints, and mixed regex and length constraints
 - Regular expression and linear arithmetic constraints only
 - Designed to isolate and evaluate the regex fragment of a string solver
- `stringfuzz-regex-transformed`
 - 11640 fuzzed industrial instances
 - Applied to security policy validation benchmarks from Amazon Web Services, handcrafted input validation cases, and the Z3str2 regular expression test cases
 - Includes regex constraints, arithmetic constraints, string-number conversion, string concatenation, word equations, and high-level operations (e.g. `indexof`, `substr`)

stringfuzz-regex-generated



stringfuzz-regex-transformed





Conclusion

- Length-aware algorithm for solving regex and length constraints
- Theoretical results on decidability and undecidability of relevant theories
- Experimental evaluation against other string solvers
- Future work: strengthening the extraction of length information from the automata we construct for regular expressions to further increase the efficacy of our method

<https://sites.google.com/site/z3strsolver>