OSTRICH Version 1.3

Taolue Chen*, Riccardo De Masellis†, Alejandro Flores-Lamas‡, Matthew Hague‡, Zhilei Han§, Denghang Hu¶
Anthony W. Lin‖, Oliver Markgraf‖, Philipp Rümmer**, Zhilin Wu††
*University of Surrey
†Uppsala University
‡University of London
§Tsinghua University
¶University of Chinese Academy of Sciences
‖Technical University of Kaiserslautern
**University of Regensburg
††State Key Laboratory of Computer Science

***Abstract*—This paper gives a high-level overview of the string solver OSTRICH version 1.3, a solver entering SMT-COMP 2023. For more details and theoretical results we refer to the full version of the paper [4] and to the website `https://github.com/uuverifiers/ostrich`.**

I. Overview

OSTRICH is a string solver designed for solving constraints that occur during program analysis. OSTRICH is built on top of the SMT solver Princess [9] and uses the BRICS Automata library [1] to handle regular expressions inside the string formulas. OSTRICH accepts constraints written using the SMT-LIB theory of strings and supports most operators of the theory. In addition, OSTRICH can handle transducers and the string reverse operation, regular expressions that include capture groups, lazy quantifiers, and anchors. OSTRICH also allows users to add their own string functions, as long as they provide an implementation of pre-image computation [4].

In SMT-COMP 2023, OSTRICH will apply several algorithms for solving string constraints, which are introduced in the next sections.

BW-Str: Backward Propagation Solver

The first algorithm implemented in OSTRICH is the background propagation procedure [4], BW-Str, which guarantees completeness for input formulas in the straight-line fragment of the theory of strings. Straight-line formulas are conjunctions of string constraints that can be represented in the form of straight-line programs S [4]:

$$S \quad ::= \quad x := f(\bar{x}) \mid \textbf{assert}(R(\bar{x})) \mid S; S$$

where f denotes n-ary string functions, for instance concatenation, replace, replace-all, reverse, or any function that can be represented by (one-way or two-way) transducers, and R is a recognisable relation represented by a collection of tuples of finite automata.

The main idea of backwards propagation is to systematically compute pre-images of regular expression constraints under the functions occurring in a straight-line program, and this way infer necessary and sufficient constraints on the input variables for the program to succeed. For a word equation like $z = x \circ y$, constraints about z are propagated to obtain constraints about x, y; a non-deterministic choice has to be made in this process which part of z is assigned to x and which to y, leading to proof branching.

In addition to backward propagation, the BW-Str solver also reasons about length constraints and number of letter occurences[1] induced by the various string operations, and ensures the consistency of such constraints.

Nielsen's transformation ([7],[5]) is applied before the backwards propagation is called, and decomposes word equations into simpler equations while imposing further length constraints. This enables BW-Str to solve also formulas outside of the straight-line fragment, although without completeness guarantees.

Finally, before and during the execution of the string solver rewriting rules are applied to cover various special cases and reduce string constraints to regular constraints to leverage the backward propagation or word equations to leverage Nielsen's transformation.

BW-Str performs well in particular for problems with complex regular expression constraints, and for problems in the straight-line fragment.

ADT-Str: List-Based Solver

The second algorithm ADT-Str implemented in OSTRICH builds on the decision procedure for algebraic data-types (ADTs) with size catamorphism implemented in Princess [6]. Algebraic data-types are used to represent strings using the standard encoding of lists with `nil` and `cons` constructors. The length of a string is computed using the build-in `size` function provided by the ADT solver, mapping every constructor term to the number constructor occurrences. Other SMT-LIB functions on strings, for instance substring, concatenation, etc., are in ADT-Str encoded using uninterpreted functions and axioms capturing the recursive definition of the string functions.

[1]Letter counting can be enabled using option `+parikh`.

Regular expression matching is implemented using Brzozowski derivatives [2].

ADT-Str is complementary to BW-Str. Although ADT-Str does not come with interesting completeness guarantees, it performs well for computing solutions of string constraints, and it can easily handle certain functions that are hard for backward propagation. Those functions include, among others, string-to-integer conversion, and functions like substring and indexof that calculate with integer offsets.

CE-STR: COST-ENRICHED SOLVER

The third algorithm CE-Str implemented in OSTRICH is based on cost-enriched finite automata (CEFAs)[3], which are automata with a cost function on each transition. The cost function encodes the string length accepted by the automaton or other integer values like the result of indexof and the integer arguments of substring. CEFA can also encode the semantics of counting operators, resulting in a more compact representation of the constraints rather than unfolding counting operators into union and concatenation operators.

The backward propagation procedure of CE-Str is inherited from BW-Str, but the automata representation of it is changed to CEFA. What's more, CE-Str combines the Parikh images[8] of propagated constraints with the input linear integer arithmetic(LIA) formulae and then solves them by the off-the-shelf LIA solvers Princess[9], guaranteeing the soundness and completeness for the straight-line fragment of the theory of strings.

CE-Str aims at solving string constraints with integer types efficiently. It is complementary to BW-Str and ADT-Str and can be used as a fallback solver when the other solvers fail to solve the constraints. With completeness guarantees, CE-Str supports more string functions like indexof and substring, avoiding rewriting them to another operation, which may break the straight-line fragment.

II. OSTRICH AT SMT-COMP 2023

We are submitting version 1.3 of OSTRICH in the single-query track divisions `QF_S`, `QF_SLIA`, `QF_SNIA`. This version is linked against Princess 2023-05-27 and the BRICS automata library 1.11-8. The submitted version of OSTRICH is configured to use the options

```
+parikh +quiet -portfolio=strings
```

Those options switch on partial Parikh reasoning, disable diagnostic output, and enable the portfolio consisting of the backward propagation-based solver (BW-Str), the ADT-based solver (ADT-Str) and the cost-enriched solver (CE-Str).

REFERENCES

[1] Brics automaton. https://www.brics.dk/automaton/index.html, accessed: 2022-06-23

[2] Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (oct 1964). https://doi.org/10.1145/321239.321249, https://doi.org/10.1145/321239.321249

[3] 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

[4] Chen, T., Hague, M., Lin, A.W., Rümmer, P., Wu, Z.: Decision procedures for path feasibility of string-manipulating programs with complex operations. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)

[5] Diekert, V.: Makanin's Algorithm. In: Lothaire, M. (ed.) Algebraic Combinatorics on Words, Encyclopedia of Mathematics and its Applications, vol. 90, chap. 12, pp. 387–442. Cambridge University Press (2002)

[6] Hojjat, H., Rümmer, P.: Deciding and interpolating algebraic data types by reduction. In: Jebelean, T., Negru, V., Petcu, D., Zaharie, D., Ida, T., Watt, S.M. (eds.) 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017. pp. 145–152. IEEE Computer Society (2017). https://doi.org/10.1109/SYNASC.2017.00033, https://doi.org/10.1109/SYNASC.2017.00033

[7] Lentin, A.: Equations dans les Monoides Libres. Gauthier-Villars, Paris (1972)

[8] Parikh, R.J.: On context-free languages. J. ACM **13**(4), 570–581 (oct 1966), https://doi.org/10.1145/321356.321364

[9] Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: International Conference on Logic for Programming Artificial Intelligence and Reasoning. pp. 274–289. Springer (2008)