OSTRICH Version 1.4

Matthew Hague*, Denghang Hu†, Anthony W. Lin‡§, Oliver Markgraf‡, Philipp Rümmer¶‖, Zhilin Wu†

*Royal Holloway, University of London, UK
†Key Laboratory of System Software (Chinese Academy of Sciences) and
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
‡Technical University of Kaiserslautern, Germany
§Max-Planck Institute for Software Systems, Germany
¶University of Regensburg, Germany
‖Uppsala University, Sweden

***Abstract*—This paper gives a high-level overview of the string solver OSTRICH version 1.4, a solver entering SMT-COMP 2024. 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 [6] 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 2024, OSTRICH will apply four algorithms for solving string constraints. The new one (RCP) is described below. The other three (BW-Str, ADT-Str and CE-Str) were already used in OSTRICH in SMT-COMP 2023 and CE-Str is updated this year; difference in CE-Str is detailed below. For more details, we refer the reader to the tool's description of OSTRICH 1.3 at https://smt-comp.github.io/2023/system-descriptions/OSTRICH.pdf. Loosely speaking, BW-Str applies Backward Propagation in combination with Nielsen's transformation and length reasoning, while ADT-Str uses a decision procedure for the theory of Algebraic Data Types (ADT) with size catamorphism.

RCP: Regular Constraint Propagation

Regular Constraint Propagation (RCP) is the newest algorithm that has been implemented in OSTRICH, based on a subset of proof rules in our paper [2]. The main goal of RCP is to *prove unsatisfiability of the input constraint*. The algorithm handles equational part with string functions like concatenation, replace, and replaceall, and regular constraints. Other string functions (e.g. reverse, one-way and two-way transducers) are permitted, so long as pre/post image of regular constraints under these functions are provided, which need not be exact, but at least overapproximate the true pre/post image. Each such formula is first converted to the following normal form by adding variables:

$$S \quad ::= \quad x = f(\bar{x}) \mid x \in L \mid S \wedge S$$

where f denotes any of the aforementioned n-ary string function. For example, the constraint $xy = yx \wedge x \in a^*ba^* \wedge y \in a^*ca^*$ will be converted to the following formula in normal form: $z = f(x,y) \wedge z = f(y,x) \wedge x \in a^*ba^* \wedge y \in a^*ca^*$, where $f(x,y) = x.y$

The main idea behind RCP is to propagate regular constraints of the form $x \in L$ to other string variables through forward and backward propagations (i.e. post/post image computation of the functions). In particular, these realize the following three proof rules. The first is the forward propagation `[Fwd-Prop]` rule:

$$\frac{\Gamma, x \in e, x = f(\bar{x}), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x_1 \in e_1 \ldots, x_n \in e_n, x = f(\bar{x})}\ C_F$$

where the side condition C_F denotes that $L(e) = f(L(e_1), \ldots, L(e_n))$. Here "forward" is to be interpreted as post image, i.e., from left to right. The second rule is backward propagation `[Bwd-Prop]`:

$$\frac{\{\Gamma, x \in e, x = f(\bar{x}), x_1 \in e_1^i, \ldots, x_n \in e_n^i\}_{i=1}^k}{\Gamma, x \in e, x = f(\bar{x})}\ C_B$$

where the side condition C_B denotes that $f^{-1}(L(e)) = \bigcup_{i=1}^k (L(e_1^i) \times \cdots \times L(e_n^i))$. Backward means the direction of the pre image of the function f. Finally, we have the `[Close]` rule:

$$\frac{}{\Gamma, x \in e_1, \ldots, x \in e_n}\ \text{if } L(e_1) \cap \cdots \cap L(e_n) = \emptyset$$

which allows us deduce unsatisfiability when the regular constraints on a variable are contradictory. Using our running example, we apply `[Fwd-Prop]` on the regular constraints $x \in a^*ba^* \wedge y \in a^*ca^*$ twice through $z = f(x,y)$ and $z = f(y,x)$ and infer, respectively, regular constraints $z \in a^*ba^*.a^*ca^*$ and $z \in a^*ca^*a^*ba^*$. These final two regular constraints allow us to conclude unsatisfiability of the formula using `[Close]`.

CE-STR: COST-ENRICHED SOLVER

In principle, the main algorithm of CE-Str (based on [3]) amounts to, as in OSTRICH 1.3, using the backward propagation procedure of BW-Str but changing the automata representation to cost-enriched finite automata (CEFAs). Nevertheless, various optimizations have been implemented in OSTRICH 1.4 to improve the performance, including:

- optimizations for reducing the sizes of CEFAs,
- optimizations for regexes with the nesting of counting operators,
- optimizations for regexes with the nesting of counting operators and a complement operator,
- optimizations for generating models of string constraints out of models of LIA formulas.

The transition of a CEFA is a tuple $(q, a, q', \vec{v})$, where q, q' are states of the CEFA, a is a character, and $\vec{v}$ is a vector of integer that represents the cost function on the transition. For example, the transition $(q, a, q', (1))$ means the CEFA can transit from state q to state q' with the register incremented by 1. We observe that the characters on transitions do not affect the values of registers in the CEFA. Moreover, the cost function $\vec{0}$ also does not affect the values of registers. Based on these observations, we drop the characters after the intersection of CEFAs and consider the cost function $\vec{0}$ as the epsilon label. In contrast, other cost functions are considered as non-epsilon labels. The transition $(q, a, q', \vec{0})$ is changed to (q, q', ϵ) and the transition $(q, a, q', \vec{v})$ is changed to $(q, q', \vec{v})$. Determinization and minimization for the transferred CEFA are then performed to reduce the size[5].

The main idea of the optimizations for regexes with the nesting of counting operators is illustrated by the constraint $x \in (a^{\{1,1000\}})^{\{1,2\}}$. If we apply the naive unfolding as last year, we will unfold $a^{\{1,1000\}}$ and obtain the constraint $x \in (a(\varepsilon + a + aa + \cdots + \underbrace{a \cdots a}_{999}))^{\{1,2\}}$. It is easy to observe that a smarter way is to unfold $^{\{1,2\}}$, instead of $^{\{1,1000\}}$. If we do this, then we get a constraint $x \in (a^{\{1,1000\}})(\varepsilon + a^{\{1,1000\}})$, whose size is much smaller, compared to $x \in (a(\varepsilon + a + aa + \cdots + \underbrace{a \cdots a}_{999}))^{\{1,2\}}$. To achieve this, we propose a score function to evaluate the number of transitions and registers produced by the unfolding of each counting operator and then unfold the counting operator producing fewer transitions and registers first.

We also propose two approximations of the complement operation on the regex with counting operators, i.e., an under-approximation and an over-approximation, and combine them into a procedure that achieves a nice balance between efficiency and precision. We compute an under-approximation of e by replacing each occurrence of the counting operators in e_1, say $e_2^{\{m,n\}}$ or $(e_2^{\{n,\infty\}})$, by e_2^*. Futhermore, for $e = \overline{e_1}$, we utilize the CEFA constructed from e_1, say $\mathcal{A}_{e_1}$, to construct a CEFA $\mathcal{B}$ of e so that $\mathcal{L}(\mathcal{B})$ is an over-approximation of $\mathcal{L}(e)$. Let $\mathcal{A}_{e_1} = (R, Q, \Sigma, \delta, I, F, \alpha)$. The main idea of the construction is explained as follows: If a string w is not accepted by $\mathcal{A}_{e_1}$, then either there are no runs of $\mathcal{A}_{e_1}$ on w, or every run of $\mathcal{A}_{e_1}$ on w stop at a non-final state or enter a final state but do not satisfy the accepting condition α. We shall construct a CEFA $\mathcal{B}$ that accepts when one of these situations occurs. Then $\overline{\mathcal{L}(\mathcal{A}_{e_1})} \subseteq \mathcal{L}(\mathcal{B})$, that is, $\mathcal{B}$ is an over-approximation of e.

In last year's submission, we generated models of string constraints out of models of LIA formulas with registers as free variables. However, this approach is not efficient enough. This year, we would like to refine this process by using the LIA formulas to obtain information about the number of occurrences of transitions instead of just the values of registers and utilize this information to guide the search process.

II. OSTRICH AT SMT-COMP 2024

We are submitting version 1.4 of OSTRICH in the single-query track divisions `QF_S`, `QF_SLIA`, `QF_SNIA`. This version is linked against Princess 2024-03-22 and the BRICS automata library 1.11-8. The submitted version of OSTRICH is configured to use the options

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

Those options disable diagnostic output and enable the portfolio consisting of regular constraint propagation (RCP), 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] Chen, T., Flores-Lamas, A., Hague, M., Han, Z., Hu, D., Kan, S., Lin, A.W., Rümmer, P., Wu, Z.: Solving string constraints with regex-dependent functions through transducers with priorities and variables. Proc. ACM Program. Lang. **6**(POPL), 1–31 (2022). https://doi.org/10.1145/3498707, https://doi.org/10.1145/3498707

[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] Hu, D., Wu, Z.: String constraints with regex-counting and string-length solved more efficiently. In: Hermanns, H., Sun, J., Bu, L. (eds.) Dependable Software Engineering. Theories, Tools, and Applications. pp. 1–20. Springer Nature Singapore, Singapore (2024)

[6] 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)