24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

String constraint solving: past, present and future

Roberto Amadini[1], Graeme Gange[2], Peter Schachte[3], Harald Søndergaard[3] and Peter J. Stuckey[2]

Abstract. String constraint solving is an important emerging field, given the ubiquity of strings over different fields such as formal analysis, automated testing, database query processing, and cybersecurity. This paper highlights the current state-of-the-art for string constraint solving, and identifies future challenges in this field.

1 Introduction

String constraint solving (or briefly, string solving) is a branch of the constraint solving field where constraints over string variables are allowed. Typical examples involve constraints on string length, (dis-)equality, concatenation, and regular expression matching.

The growing interest in string solving has emerged in application domains where string processing plays a central role such as test-case generation, software analysis and verification, model checking, web security, and database query processing [14, 15, 10, 7, 4].

In particular, the widespread interest in *cybersecurity* has given new impulse to research in string solving. Strings are often the silent carriers of software vulnerabilities; for example, they are used in various forms of injection attacks. In 2019, the first workshop on String Constraints and Applications has been organised [12].

Here we briefly review the literature on string constraint solving. We summarize the state-of-the art in this field and conclude by giving some of the possible future challenges.

2 State of the art

We formalise the general concept of string constraint solving by instantiating the definition of *constraint satisfaction problem* (CSP). A CSP is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where: $\mathcal{X} = \{x_1, \dots, x_n\}$ are the variables; $\mathcal{D} = \{D(x_1), \dots, D(x_n)\}$ the domains, where for $i = 1, \dots, n$ each $D(x_i)$ is a set of values that x_i can take; $\mathcal{C}$ are the constraints defined over the variables of $\mathcal{X}$.

Given a finite alphabet Σ, a *string constraint satisfaction problem* is a CSP containing $k > 0$ *string variables* $\{w_1, \dots, w_k\} \subseteq \mathcal{X}$ such that $D(w_i) \subseteq \Sigma^*$ for $i = 1, \dots, k$ and at least one constraint of $\mathcal{C}$ involves a string variable. The goal of string solving is to find a solution (or prove the unsatisfiability) of a given string CSP, i.e., an assignment $\xi \in D(x_1) \times \dots \times D(x_n)$ of domain values to the corresponding variables that satisfies all of the constraints of $\mathcal{C}$—in particular, ξ assigns a string of Σ^* to each string variable w.

Several variants of string CSPs can be defined. For example, we can get an optimisation problem by adding an objective function. An important classification for string CSPs concerns the *boundedness* of string variables: we talk about bounded-length, or simply bounded, string solving when there is an upper bound $\lambda > 0$ on the string length of each variable. When not explicit, deciding the value of λ may be non-trivial: a too small λ may exclude feasible solutions, while a too big λ may significantly slow down the solving process.

All the approaches we are aware for string constraint solving can be grouped in the following three categories:

- *Automaton-based approaches:* These approaches rely on automata to handle string variables and constraints (e.g., [30, 22, 19]). They can handle unbounded-length strings and represent infinite sets of strings precisely. However, automata typically have performance issues due to state explosion and the integration with other domains (integers in particular).
- *Word-based approaches:* These approaches solve systems of *word equations*, possibly enriched with other constraints (e.g., string length or regular membership constraints). They mainly rely *satisfiability modulo theory* (SMT) solvers to tackle such constraints [23, 13, 9, 2, 29, 18, 1, 11]. SMT string solvers handle unbounded strings and can rely on several already defined theories, but unfortunately most of them are incomplete and suffer from the performance issues of the underlying DPLL(T) paradigm [16].
- *Unfolding-based approaches:* These approaches basically unfold each string variable x into $k > 0$ contiguous variables representing (sets of) characters of x. For example, x can be mapped into k integer variables [28] or to bit-vectors [20, 27]. The approach, well-suited for *constraint programming* (CP) solvers, cannot deal with unbounded-length strings and may be inefficient if the length bound λ is large. To overcome the latter issue, a recently introduced CP approach, which can be seen as a "lazy" unfolding, devised the *dashed string* abstraction [6].

Several solvers have been proposed for string constraint solving. To the best of our knowledge, at present the most effective and "general purpose" (i.e., solvers not tailored to solve a specific string CSP) are CVC4 [23], G-STRINGS [6], and Z3 [13].

CVC4 and Z3 are SMT solvers supporting the theory of word-equations with additional constraints (e.g., string length or regular expressions). Z3 comes in two flavors: a version using the theory of sequences and one using the string solver Z3STR3 [9].

G-STRINGS is a CP solver integrating string solving capabilities into the GECODE solver [17]. It is based on a concept of "dashed strings"—simplified regular expressions able to express the concatenation of finite sets (blocks) of strings. G-STRINGS maintains a domain for each string variable (in the form of a dashed string) and defines a *constraint propagator* for each string constraint [5, 6]. The propagators generally rely on a block refinement principle based on the "sweep" algorithm for scheduling problems [3].

Among the string constraints supported by all of the above solvers we mention (dis-)equality, concatenation, length, substring, find and replace, channelling between integers and strings, and regular expressions operations. In addition, G-STRINGS can handle lexico-

[1] University of Bologna, Bologna, Italy
[2] Monash University, Caulfield East Vic. 3145, Australia
[3] The University of Melbourne, Melbourne Vic. 3010, Australia

graphic ordering [5] and character counting.

3 Future challenges

We identify four main challenges for string constraint solving:

- *Extend string solving.* Support for string solving is fairly recent, and therefore there are several string constraints that no solver is able to handle. Among them, we mention complex (extended) regular expressions such as *back-references*, *lookaheads/lookbehinds* or *greedy matching*. Complex (extended) regular expressions occur very frequently in JavaScript—the *de facto* language for web programming [25]. Such operations have to be addressed properly to significantly improve the analysis of web programming.
- *Improve string solving.* Improving the efficiency of string solvers is of course very welcome. This means to devise new solving algorithms and search heuristics. At present, SMT solvers tend to fail with long-length strings, while CP solvers may struggle to prove unsatisfiability. In particular, an interesting—and far from trivial—challenge for CP string solvers concerns the study of string solving and *clause learning*, a powerful technique that dramatically improved the performance of modern solvers [26].
- *Combine string solving.* Constraint solvers have disparate nature and often display uneven performance across different (types of) problem instances. Over the last years, plenty of evaluations have shown that a *portfolio* of different solvers can significantly outperform a single, arbitrarily efficient solver. A promising future challenge is therefore the definition of portfolios of string solvers [21], possibly running simultaneously and collaboratively (e.g., through information exchange between solvers). A preliminary study on the dynamic symbolic execution of JavaScript has shown promising results [4].
- *Utilize string solving.* The utilization and proliferation of solvers and related tools is the final goal for string constraint solving. This is clearly subordinated to the implementation of the above steps. Software verification, testing, model checking and cybersecurity seem to be a suitable fertile ground for the dissemination of string solving. Some approaches already integrate string solving capabilities into their dynamic symbolic execution frameworks [24, 4]. However, string solving may also be useful in other fields such as bioinformatics [8].

ACKNOWLEDGEMENTS

This work is supported by the Australian Research Council (ARC) through Linkage Project Grant LP140100437 and Discovery Early Career Researcher Award DE160100568.

REFERENCES

[1] P. A. Abdulla, M. F. Atig, Y. Chen, B. P. Diep, L. Holík, A. Rezine, and P. Rümmer. Flatten and conquer: A framework for efficient analysis of string constraints. In *PLDI 2017*, pages 602–617, 2017.

[2] P. A. Abdulla, M. F. Atig, Y.-F. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. Norn: An SMT solver for string constraints. In D. Kroening and C. Păsăreanu, editors, *27th CAV, Part I*, volume 9206 of *LNCS*, pages 462–469. Springer, 2015.

[3] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[4] R. Amadini, M. Andrlon, G. Gange, P. Schachte, H. Søndergaard, and P. J. Stuckey. Constraint programming for dynamic symbolic execution of JavaScript. In *16th CPAIOR*, volume 11494 of *LNCS*, pages 1–19. Springer, 2019.

[5] R. Amadini, G. Gange, and P. J. Stuckey. Propagating *Lex*, *Find* and *Replace* with dashed strings. In *15th CPAIOR*, volume 10848 of *LNCS*, pages 18–34. Springer, 2018.

[6] R. Amadini, G. Gange, and P. J. Stuckey. Sweep-based propagation for string constraint solving. In *32nd AAAI Conf. Artificial Intelligence*, pages 6557–6564. AAAI Press, 2018.

[7] R. Amadini, A. Jordan, G. Gange, F. Gauthier, P. Schachte, H. Søndergaard, P. J. Stuckey, and C. Zhang. Combining string abstract domains for JavaScript analysis: An evaluation. In *TACAS 2017*, pages 41–57. Springer, 2017.

[8] P. Barahona and L. Krippahl. Constraint programming in structural bioinformatics. *Constraints*, 13(1-2):3–20, 2008.

[9] M. Berzish, V. Ganesh, and Y. Zheng. Z3str3: A string solver with theory-aware heuristics. In *17th FMCAD*, pages 55–59. FMCAD, 2017.

[10] P. Bisht, T. L. Hinrichs, N. Skrupsky, and V. N. Venkatakrishnan. WAPTEC: Whitebox analysis of web applications for parameter tampering exploit construction. In *18th CCS*, pages 575–586. ACM, 2011.

[11] T. Chen, M. Hague, A. W. Lin, P. Rümmer, and Z. Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *PACMPL*, 3(POPL):49:1–49:30, 2019.

[12] L. D'Antoni, A. W. Lin, and P. Rümmer. Meeting on string constraints and applications, 2019. `https://mosca19.github.io/`.

[13] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *14th TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[14] M. Emmi, R. Majumdar, and K. Sen. Dynamic test input generation for database applications. In *ISSTA 2007*, pages 151–162. ACM, 2007.

[15] G. Gange, J. A. Navas, P. J. Stuckey, H. Søndergaard, and P. Schachte. Unbounded model-checking with interpolation for regular language constraints. In *19th TACAS*, volume 7795 of *LNCS*, pages 277–291. Springer, 2013.

[16] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. O. a, and C. Tinelli. DPLL(T): Fast decision procedures. In *16th CAV*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.

[17] Gecode Team. Gecode: Generic constraint development environment, 2016. Available at `http://www.gecode.org`.

[18] L. Holík, P. Janku, A. W. Lin, P. Rümmer, and T. Vojnar. String constraints with concatenation and transducers solved efficiently. *PACMPL*, 2(POPL):4:1–4:32, 2018.

[19] P. Hooimeijer and W. Weimer. StrSolve: Solving string constraints lazily. *Automated Software Engineering*, 19(4):531–559, 2012.

[20] A. Kieżun, V. Ganesh, S. Artzi, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Trans. Software Engineering and Methodology*, 21(4):article 25, 2012.

[21] L. Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, volume 10101 of *LNAI*, pages 149–190. Springer, 2016.

[22] G. Li and I. Ghosh. PASS: String solving with parameterized array and interval automaton. In *9th Int. Haifa Verification Conf.*, volume 8244 of *LNCS*, pages 15–31. Springer, 2013.

[23] T. Liang, A. Reynolds, C. Tinelli, C. Barrett, and M. Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In *26th CAV*, volume 8559 of *LNCS*, pages 646–662. Springer, 2014.

[24] B. Loring, D. Mitchell, and J. Kinder. ExpoSE: Practical symbolic execution of standalone JavaScript. In *24th Int. Symp. Model Checking of Software (SPIN'17)*, pages 196–199. ACM, 2017.

[25] B. Loring, D. Mitchell, and J. Kinder. Sound regular expression semantics for dynamic symbolic execution of JavaScript. In *40th PLDI*, pages 425–438. ACM, 2019.

[26] O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[27] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A symbolic execution framework for JavaScript. In *Proc. 2010 IEEE Symp. Security and Privacy*, pages 513–528. IEEE Comp. Soc., 2010.

[28] J. D. Scott, P. Flener, J. Pearson, and C. Schulte. Design and implementation of bounded-length sequence variables. In *14th CPAIOR*, volume 10335 of *LNCS*, pages 51–67. Springer, 2017.

[29] M. Trinh, D. Chu, and J. Jaffar. Model counting for recursively-defined strings. In *29th CAV*, volume 10427 of *LNCS*, pages 399–418. Springer, 2017.

[30] F. Yu, M. Alkhalaf, and T. Bultan. Stranger: An automata-based string analysis tool for PHP. In *16th TACAS*, volume 6015 of *LNCS*, pages 154–157. Springer, 2010.