String constraint solving: past, present and future

Roberto Amadini\textsuperscript{1}, Graeme Gange\textsuperscript{2} Peter Schachte\textsuperscript{3} Harald Søndergaard\textsuperscript{3} and Peter J. Stuckey\textsuperscript{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 tuple \( \langle X, D, C \rangle \) where: \( X = \{ x_1, \ldots, x_n \} \) are the variables; \( D = \{ D(x_1), \ldots, D(x_n) \} \) the domains, where for \( i = 1, \ldots, n \) each \( D(x_i) \) is a set of values that \( x_i \) can take; \( C \) are the constraints defined over the variables of \( X \).

Given a finite alphabet \( \Sigma \), a string constraint satisfaction problem is a CSP containing \( k > 0 \) string variables \( \{ w_1, \ldots, w_k \} \subseteq X \) such that \( D(w_i) \subseteq \Sigma^* \) for \( i = 1, \ldots, k \) and at least one constraint of \( C \) 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 \( \xi \in D(x_1) \times \cdots \times D(x_n) \) of domain values to the corresponding variables that satisfies all of the constraints of \( C \)—in particular, \( \xi \) assigns a string of \( \Sigma^* \) 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 \( \lambda \) may be non-trivial: a too small \( \lambda \) may exclude feasible solutions, while a too big \( \lambda \) 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 on 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 \( \lambda \) 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-
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, lookahead/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


