Constraint programming and (dashed) string solving

Roberto Amadini

Department of Computer Science and Engineering, University of Bologna, Italy

Meeting On String Constraints and Applications
July 17th, 2023. Paris, France.

String analysis

- Interest in string analysis active and growing over last decade
  - Test-Case Generation
  - Program Analysis
  - Model Checking
  - Web Security
- Static string analysis: over-approximation of string computations
  - E.g., Abstract Interpretation with string abstract domains
- Dynamic string analysis: under-approximation of string computations
  - E.g., (Dynamic) Symbolic Execution (DSE) of programs with strings
    - We need (string) constraint solving to solve path condition constraints

String constraint solving

- String constraint solving (SCS) = solving combinatorial problems with string variables and constraints on given alphabet Σ
  - Foundations lay in theory of automata and combinatorics on words
  - Word equations $L = R$ with $L, R \in (\Sigma \cup \mathit{Vars})^*$ are central in SCS
  - General FOL theory undecidable, quantifier-free theory decidable (Makanin, 1977)

- We can classify string variables into:
  - fixed-length: given $\lambda \in \mathbb{N}$, only take values in $\{w \in \Sigma^* \mid |w| = \lambda\}$
  - bounded-length: given $\lambda \in \mathbb{N}$, only take values in $\{w \in \Sigma^* \mid |w| \leq \lambda\}$
  - unbounded-length: they can take any value in Σ^*

- String constraints: length, concatenation, regular, find/replace(-all), lexicographic ordering, conversion string $\leftrightarrow$ numbers

String constraint solvers

- We can classify string solvers into:
  - automata-based: string variables represented by automata, string constraints mapped to automata operations
  - word-based: natively handle theory of word-equations + extensions
  - unfolding-based: reduce to sequences of $k \geq 0$ variables of type T
  - No sharp distinction, efficient SOTA solvers are hybrid

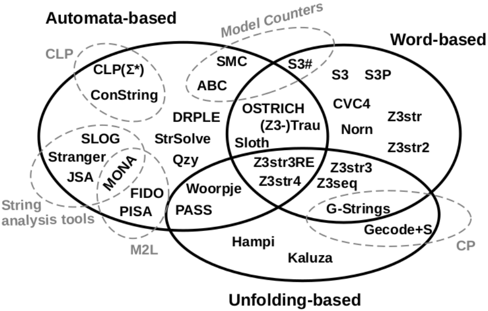


Figure from [Amadini, 2021]

String constraint solvers

- Most SOTA "general-purpose" string solvers are SMT-based
  - SMT-LIB developed a theory of unicode strings
- Some Constraint Programming (CP) proposals too
- From CP perspective, SCS = solving CSP/COP with bounded-length string variables: a maximum string length λ is fixed
  - Goal: assigning a consistent string literal to each string variable
- CP proposals are mainly unfolding-based:
  - Bounded-length sequence (B.L.S.) variables [Scott *et al.*, 2017]
    - String variable x unfolded into max $|x|$ integer variables c_i^x = i-th char of x (possibly empty), plus 1 integer variable n_x for $|x|$
  - Dashed string (D.S.) variables [Amadini *et al.*, 2020]

Dashed strings

- Dashed strings: simplified regular expressions representing set of strings in a more compact way w.r.t. B.L.S.
  - Inspired by Bricks abstract domain by [Costantini *et al.*, 2015]

- D.S. enable a more "lazy" unfolding w.r.t. B.L.S.: blocks group together "similar regions" of the target string
  - If no clue on length upper bound, B.L.S. needs $\lambda + 1$ integer variables

- E.g., D.S. $\{b, c\}^{1,1}\{a\}^{0,1}\{d\}^{1,2}$ denotes all the strings:
  - Starting with 1 b's or c's, followed by 0 or 1 a's, followed by 1 or 2 d's
  - . . . i.e., the set of strings $\{bd, bdd, bad, badd, cd, cdd, cad, cadd\}$

- With B.L.S. representation it would be $\{b, c\}\{a, d\}\{\epsilon, d\}\{\epsilon, d\}$, denoting $\{ba, bd, ca, cd, bad, bdd, cad, cdd, badd, bddd, cadd, cddd\}$
  - More blocks, less precise abstraction

Dashed strings

- Formally, a D.S. is a concatenation of blocks $S_1^{l_1,u_1} S_2^{l_2,u_2} \cdots S_k^{l_k,u_k}$ with $S_i \subseteq \Sigma$ and $0 \leq l_i \leq u_i \leq \lambda$
- Each block $S_i^{l_i,u_i}$ denotes $\gamma(S_i^{l_i,u_i}) = \{w \in S_i^* \mid l_i \leq |w| \leq u_i\}$
- If $\mathbb{S} = \{w \in \Sigma^* \mid |w| \leq \lambda\}$, each D.S. $X = S_1^{l_1,u_1} S_2^{l_2,u_2} \cdots S_k^{l_k,u_k}$ denotes: $\gamma(X) = \left(\gamma(S_1^{l_1,u_1}) \cdot \gamma(S_2^{l_2,u_2}) \cdot \cdots \cdot \gamma(S_k^{l_k,u_k})\right) \cap \mathbb{S}$
- E.g, if $\Sigma = \{a, b, c\}$ and $\lambda = 3$, $X = \{a\}^{1,2}\{b, c\}^{0,2}$ denotes $\gamma(X) = \{a, ab, ac, abb, abc, acb, acc, aa, aab, aac\}$
  - $aabb, aabc, aacb, aacc \notin \gamma(X)$ because they have length $> \lambda$

Graphical interpretation

- Each block $S_i^{l_i,u_i}$ can be seen as:
  - a continuous segment of length l_i (the mandatory part of the block), followed by
  - a dashed segment of length $u_i - l_i$ (the optional part of the block)
- E.g., graphical representation of $X = \{\texttt{B,b}\}^{1,1}\{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}\{\texttt{!}\}^{0,3}$

- $\gamma(X) = \{\texttt{Bom}, \texttt{bom}, \texttt{Boom}, \texttt{boom}, \ldots, \texttt{Booom!!!}, \texttt{booom!!!}\}$

Dashed string equation

- Unfortunately, D.S. cannot precisely represent all the $W \subseteq \Sigma^*$

- E.g., if $W = \{ab, ba\}$ there is no a "best representation" X for W s.t. $\gamma(X) = W$
  - We may take $\{a\}^{0,1}\{b\}^{1,1}\{a\}^{0,1}$, $\{b\}^{0,1}\{a\}^{1,1}\{b\}^{0,1}$ or $\{a, b\}^{2,2}$ as over-approximations: $\gamma(X) \supset W$

- We need workarounds to find "good enough" approximations and domains' refinement

- Given D.S. X, Y we define the D.S. equation between X and Y as a refinement operation $\textsc{Equate}(X, Y)$ s.t.
  - if $\textsc{Equate}(X, Y) = \bot$, then $\gamma(X) \cap \gamma(Y) = \emptyset$
  - if $\textsc{Equate}(X, Y) = (X', Y')$, then $\gamma(X') \subseteq \gamma(X)$, $\gamma(Y') \subseteq \gamma(Y)$, and $\gamma(X) \cap \gamma(Y) = \gamma(X') \cap \gamma(Y')$

Dashed string equation

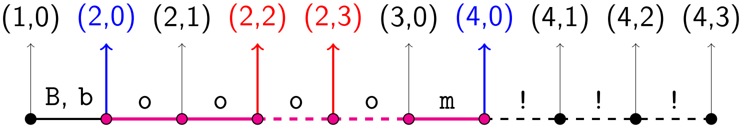
- Most of D.S. propagators we propose are based on sweep-based equation algorithms matching blocks against portions of D.S.

- Suitable PUSH and STRETCH operations are used to find the earliest/latest start/end positions where a block can match a D.S.
    - PUSH: consumes the least characters of a block, can "jump"
    - STRETCH: consumes the most characters of a block
    - ESP...LEP $=$ feasible region $\supseteq$ mandatory region $=$ LSP...EEP
    - If there is no feasible match, $\perp$ is returned

- Matching regions are then used to possibly refine the blocks
    - A normalization is used to remove spurious configurations
    - E.g., $\textsc{Norm}(\{a\}^{0,2}\,\emptyset^{1,5}\,\{a\}^{1,1}) = \{a\}^{1,3}$

Dashed string equation

- E.g., matching $B = \{\texttt{o}, \texttt{m}, \texttt{g}\}^{2,6}$ vs. $X = \{\texttt{B,b}\}^{1,1}\{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}\{\texttt{!}\}^{0,3}$

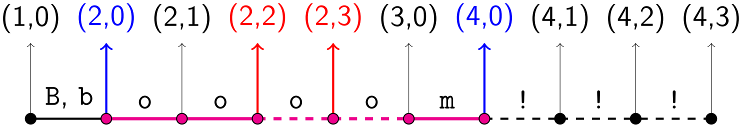


- The earliest start position of B is $(2, 0)$
- The latest start position of B is $(2, 3)$
- The earliest end position of B is $(2, 2)$
- The latest end position of B is $(4, 0)$

- Feasible region = ESP...LEP = $X[(2, 0), (4, 0)] = \{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}$
- Mandatory region = LSP...EEP = $X[(2, 3), (2, 2)] = \emptyset^{0,0}$

Dashed string equation

- E.g., matching $B = \{\mathtt{o}, \mathtt{m}, \mathtt{g}\}^{2,6}$ vs. $X = \{\mathtt{B},\mathtt{b}\}^{1,1}\{\mathtt{o}\}^{2,4}\{\mathtt{m}\}^{1,1}\{\mathtt{!}\}^{0,3}$



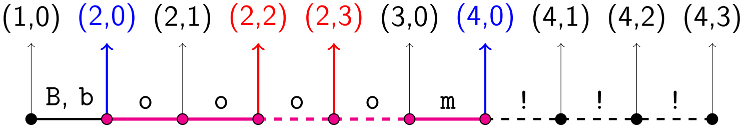
- Feasible region = ESP...LEP = $X[(2,0),(4,0)] = \{\mathtt{o}\}^{2,4}\{\mathtt{m}\}^{1,1}$: B must match X within this region

- Mandatory region = LSP...EEP = $X[(2,3),(2,2)] = \emptyset^{0,0}$: no precise information about which blocks are surely matched by B

- What we know is that ≥ 2 blocks and ≤ 6 blocks of the feasible region must be matched by B

Dashed string equation

- E.g., matching $B = \{\texttt{o}, \texttt{m}, \texttt{g}\}^{2,6}$ vs. $X = \{\texttt{B},\texttt{b}\}^{1,1}\{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}\{\texttt{!}\}^{0,3}$



- Feasible region = ESP...LEP = $X[(2,0),(4,0)] = \{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}$
- Mandatory region = LSP...EEP = $X[(2,3),(2,2)] = \emptyset^{0,0}$

- We cannot in general refine B into $\{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}$: there might be blocks before/after B matching $X[(2,0),(4,0)]$

- Surely we can CRUSH the feasible region $\{\texttt{o}\}^{2,4}\{\texttt{m}\}^{1,1}$ into $\{\texttt{o}, \texttt{m}\}^{3,5}$ and refine B into $(\{\texttt{o}, \texttt{m}, \texttt{g}\} \cap \{\texttt{o}, \texttt{m}\})^{2,\min(6,5)} = \{\texttt{o}, \texttt{m}\}^{2,5}$

Dashed string equation

- The worst-case complexity of finding matching positions is linear in the number of blocks of X and Y
  - $\mathsf{ESP}_{k+1} \equiv \mathsf{EEP}_k$, $\mathsf{LSP}_{k+1} \equiv \mathsf{LEP}_k$
  - The refinement might be quadratic, but it's a very rare case
- We proved that our approach is sound
  - Its completeness is still an open issue: if $\gamma(X) \cap \gamma(Y) = \emptyset$, we have no proof that $\textsc{Equate}(X, Y) = \bot$
- We built several propagators for string constraints on top of $\textsc{Equate}$
  - (Dis-)equality, reified equality, concatenation, . . .

Propagators EQUATE-based

- $x = y$
- $x \neq y$
- $b \iff x = y$
- $z = x \cdot y$
- $y = x^n$
- $y = x^{-1}$
- $y = x[i..j]$
- FIND
- REPLACE
- REPLACE-ALL
- $\ldots$

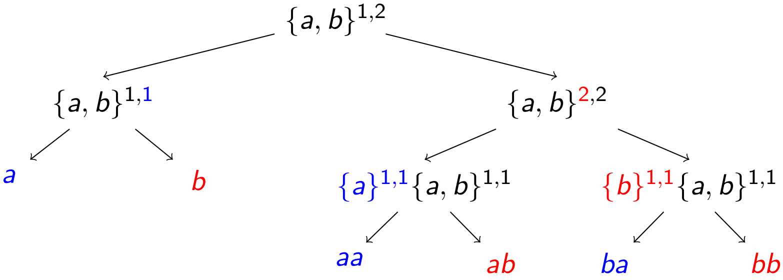
Propagators not EQUATE-based

- $n = |x|$
- $x \prec y$, $x \preceq y$, $x \succeq y$, $x \succ y$
- $x \in \mathcal{L}(R)$
- $i = \mathrm{match}(x, \rho)$

Branching

- Often we need to branch on string variables to get a feasible solution

- We first fix the length of a string variable, then the cardinality of one of its blocks, and finally the base of that block



$\{a, b\}^{1,2}$

$\{a, b\}^{1,1}$

$\{a, b\}^{2,2}$

a

b

$\{a\}^{1,1}\{a, b\}^{1,1}$

$\{b\}^{1,1}\{a, b\}^{1,1}$

aa

ab

ba

bb

Implementation

- We implemented string variables, propagators and branchers into G-STRINGS
  - Experimental extension of GECODE solver, written in C++
- We developed a MiniZinc interface to ease the modeling of SCS problems, and a compiler MiniZinc→SMT-LIB
  - Most state-of-the-art string solvers are SMT-based
- We performed several evaluations over different benchmarks with good results, especially with long strings and big regex
  - E.g., *StringFuzz* benchmarks
- G-STRINGS development a bit quiet now :-(

Conclusions

- We presented a CP approach for SCS based on dashed strings
- Possible extensions
  - New string constraints
  - New branching heuristics
  - Clause learning
  - Portfolios of string solvers
  - . . .
- How to involve students / companies? :)

Thanks for your attention!

References

[Amadini *et al.*, 2020] Roberto Amadini, Graeme Gange, and Peter J. Stuckey.
Dashed strings for string constraint solving.
Artif. Intell., 289:103368, 2020.

[Amadini, 2021] Roberto Amadini.
A survey on string constraint solving.
ACM Comput. Surv., 55(1), nov 2021.

[Costantini *et al.*, 2015] Giulia Costantini, Pietro Ferrara, and Agostino Cortesi.
A suite of abstract domains for static analysis of string values.
Software: Practice and Experience, 45(2):245–287, 2015.

[Scott *et al.*, 2017] Joseph D. Scott, Pierre Flener, Justin Pearson, and Christian Schulte.
Design and implementation of bounded-length sequence variables.
In *CPAIOR*, volume 10335 of *LNCS*, pages 51–67. Springer, 2017.