Formal Methods
in Computer-Aided Design
Prague, 16-18.10.2024

Word Equations as Abstract Domain for String Manipulating Programs

Antonina Nepeivoda
Program Systems Institute of RAS
a_nevod@mail.ru

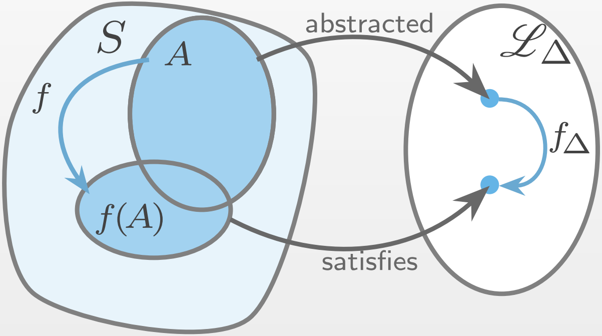
Abstract Program Properties

- We are interested in satisfiability of program properties wrt some predicate set Δ over program data. The set Δ is to be expressed in a chosen language $\mathscr{L}_\Delta$.
- Then any program trace can be considered in terms of abstracted structures wrt preserving properties in Δ, i.e. abstracted to $\mathscr{L}_\Delta$.

- Checking division by $0 \Rightarrow$ $\mathscr{L}_\Delta = \{\mathtt{IsZero}, \mathtt{IsNonZero}, \mathtt{Unknown}, \mathtt{Error}\}$.
- Checking overflows in arithmetics $\Rightarrow \mathscr{L}_\Delta$ can be an interval predicate set $\{(x \in (a, b)) \mid a, b \in \mathbb{R} \cup \{-\infty, \infty\}\} \cup \{\mathtt{Error}\}$.

Function Lifting

- Every function $f : S \to S$ in the program data domain is to be lifted to the abstract values domain.
- The lifted image of f (denoted f_Δ) must respect the original function f properties.



Galois Connection in Abstract Interpretation

Let S and Δ be sets of concrete and abstract data values. We say that $a \propto \rho$ iff $a \in S$ satisfies the predicate $\rho \in \Delta$.

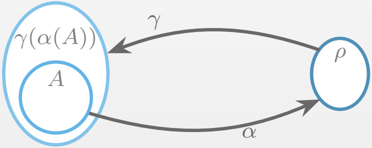
The two functions are crucial to analyse the abstract properties:

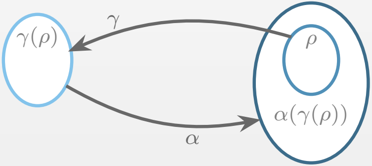
- Abstraction $\alpha : 2^S \to \Delta$;

  $\alpha(A) =$ most specific predicate ρ such that $\forall a\,(a \in A \Rightarrow a \propto \rho)$;

- Concretisation: $\gamma : \Delta \to 2^S$;

  $\gamma(\rho) = \{a \in S \mid a \propto \rho\}$.





$A \subseteq \gamma(\alpha(A))$

$\rho \overset{???}{\preceq} \alpha(\gamma(\rho))$

Such a relation between any two sets in called a Galois connection.

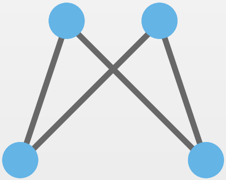
Partial Order on Δ and Lattice Axioms

Δ is equipped by partial order $\preceq$, and a pair of functions $\alpha : 2^S \to \Delta$, $\gamma : \Delta \to 2^S$ over-approximate elements of 2^S w.r.t. $\preceq$ and $\subseteq$.

- $A \subseteq \gamma(\alpha(A))$;
- $\rho \preceq \alpha(\gamma(\rho))$.

Collecting semantics

Given $\rho_1, \rho_2 \in \mathscr{L}_\Delta$, a joined value ρ' s.t. $\rho_1 \preceq \rho'$ and $\rho_2 \preceq \rho'$ is to be unique $\Rightarrow$ ρ' is a unique least upper bound of ρ_1, ρ_2.



An ordering with multiple upper bounds cannot produce an unambiguous semantics of the joined values.

Partial Order on Δ and Lattice Axioms

Δ is equipped by partial order $\preceq$, and a pair of functions $\alpha : 2^S \to \Delta$, $\gamma : \Delta \to 2^S$ over-approximate elements of 2^S w.r.t. $\preceq$ and $\subseteq$.

- $A \subseteq \gamma(\alpha(A))$;
- $\rho \preceq \alpha(\gamma(\rho))$.

In a lattice, every two elements of Δ have supremums ($\vee$) and infinums ($\wedge$) w.r.t. the partial order $\preceq$.

Lattice Axioms

- $(x \vee (x \wedge y) = x)$ & $(x \wedge (x \vee y) = x)$;
- $(x \vee y = y \vee x)$ & $(x \wedge y = y \wedge x)$;
- $(x \vee (y \vee z) = (x \vee y) \vee z)$ & $(x \wedge (y \wedge z) = (x \wedge y) \wedge z)$.

Existing Lattices for Strings

Let Σ be a letter alphabet, $S \subseteq \Sigma^*$.

- The simplest lattice over predicates $\{X = \xi \mid \xi \in \Sigma^*\}$ (denoted $\mathcal{L}att_{\mathrm{Eq}}$). Finite height, imprecise.

mapping to commutative algebra

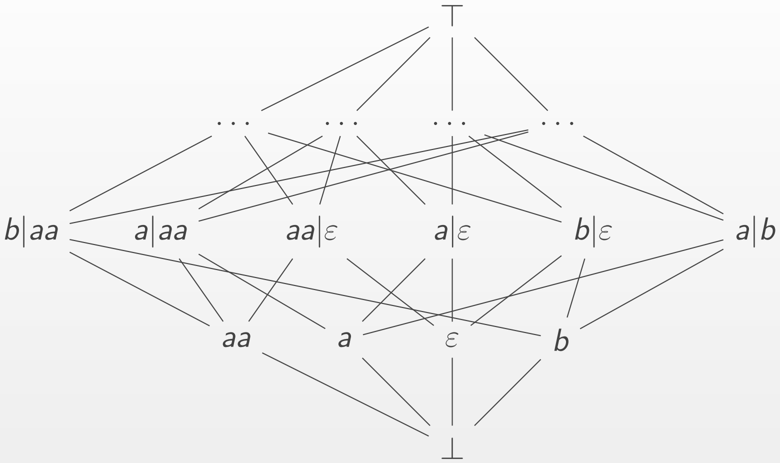
- Lattice tracking string lengths: $\{|X| = n \mid n \in \mathbb{N}\}$.
- Lattice tracking letter occurrences in strings: $\{(|X|_q > 0) \mid q \in \Sigma\}$.

incomplete

- Lattice tracking prefixes and suffixes of strings: $\{\exists \tau_1, \tau_2 (X = \xi_1 \tau_1) \,\&\, (X = \tau_2 \xi_2) \mid \xi_i \in \Sigma^*\}$.
- Lattice checking membership in regular languages: $\{X \in \tau \mid \tau \text{ is a regular expression}\}$.

- Lattice tracking program-specific predicates (JSAI, Kashyap et al, 2014). Desirable to be parameterizable with predicates.

Regular Expressions Lattice



requires widening $\left\{\right.$ Both infinite ascending chains (e.g. a, $a|a^2$,. . .) and infinite descending chains (e.g. a^*, $(a^2)^*$,. . .).

The Research Questions Arise

- Find an expressible string predicates set, forming a lattice of string abstract values:
  - being complete (and, moreover, finite-height);
  - extending the trivial lattice $\mathcal{L}att_{\mathrm{Eq}}$;
  - capturing non-commutative string properties.
- Improve flexibility of the given predicate set using lattice operations.

We suggest a word equation language as a candidate for such a set of abstract values.

Word Equations by Example

Given two alphabets:

- Σ is a set of constant (e.g. lowercase latin) characters;
- $\mathcal{V}$ is a set of variables (e.g. capitalized latin characters).

Consider two constant words:

$$\mathsf{aababaa} \qquad \mathsf{aababaa}$$

Word Equations by Example

Given two alphabets:

- Σ is a set of constant (e.g. lowercase latin) characters;
- $\mathcal{V}$ is a set of variables (e.g. capitalized latin characters).

Replace some of occurrences of their subwords with variables:

$$aababaaba \qquad aababababaa$$

$$a^2 \mapsto X, \quad a^2ba \mapsto Y, \quad b \mapsto Z$$

$$XZabY \qquad YbaZX$$

Word Equations by Example

Given two alphabets:

- Σ is a set of constant (e.g. lowercase latin) characters;
- $\mathcal{V}$ is a set of variables (e.g. capitalized latin characters).

Insert the equality sign between the resulted words:

$$aababaaba \qquad aababababaa$$

$$a^2 \mapsto X, \quad a^2ba \mapsto Y, \quad b \mapsto Z$$

$$XZabY \qquad YbaZX$$

$$XZabY \quad = \quad YbaZX$$

Word Equations by Example

Given two alphabets:

- Σ is a set of constant (e.g. lowercase latin) characters;
- $\mathcal{V}$ is a set of variables (e.g. capitalized latin characters).

$$aababaaba \qquad aabababaa$$

$$a^2 \mapsto X, \quad a^2ba \mapsto Y, \quad b \mapsto Z$$

$$XZabY \quad = \quad YbaZX$$

We have obtained a word equation.

A solution of the equation is the following substitution:

$$X := a^2, \quad Y := a^2ba, \quad Z := b$$

Word Equations by Example

Given two alphabets:

- Σ is a set of constant (e.g. lowercase latin) characters;
- $\mathcal{V}$ is a set of variables (e.g. capitalized latin characters).

$$aababaaba \qquad aabababaa$$

$$a^2 \mapsto X, \quad a^2ba \mapsto Y, \quad b \mapsto Z$$

$$XZabY \quad = \quad YbaZX$$

We have obtained a word equation.

A solution of the equation is the following substitution:

$$X := a^2, \quad Y := a^2ba, \quad Z := b$$

Another solution of the equation :

$X := \varepsilon$, $Y := a$, $Z := \varepsilon$, where ε is the empty word.

Word Equations

Given a finite alphabet Σ of constant characters and an alphabet $\mathcal{V}$ of variables. Let ε stand for the empty word.

Definition

A word equation is an equation of the form $\Psi = \Phi$, where $\Psi, \Phi \in \{\Sigma \cup \mathcal{V}\}^*$. One may see the equation as a pair $\langle \Phi, \Psi \rangle$.

Given a word equation $\Psi = \Phi$, where $\Psi, \Phi \in \{\Sigma \cup \mathcal{V}\}^*$. A solution of the equation is a morphism $\sigma : \{\Sigma \cup \mathcal{V}\}^* \to \Sigma^*$ s.t. $\sigma(\Psi) = \sigma(\Phi)$, where the morphism σ respects the concatenation operation, and $\forall \xi \in \Sigma.\ \sigma(\xi) = \xi$.

Word Equations in Program Analysis (Simplest Examples)

How do word equations naturally arise in the context of program analysis?

The variables X, X_1, Y, U, Z range over strings. Their values may be completely unknown or partially known in compile (analyse) time.

$$X = UbaYabZ$$

```
if ( includes(X, 'ba' + Y + 'ab') )
  then { ...... }
  else { ...... }
```

$$X = aX_1 = X_1 a$$

```
while ( startsWith(X,'a') && endsWith(X,'a') ) {
  X := substring(X, 1, length(X))
  if is_empty(X) return true;
}
```

Word Equations in Program Analysis

Example source: P.A. Abdulla, M.F. Atig, Y. Chen, B.P. Diep, L. Holik, A. Rezine, Ph. Rummer: Flatten and conquer: a framework for efficient analysis of string constraints, PLDI, 2017, ACM, pp. 602–617.

Initial program

```
main(X,Y,Z)   =
      eq(g(Z)++X, 'a'++g(Z)++Y);
g('i'++X)     =   'a'++g(X)++'b';
g('s'++X)     =   g(X)++'b';
g(ε)          =   ε;
eq(X, X)      =   true;
eq(X, Y)      =   false;
```

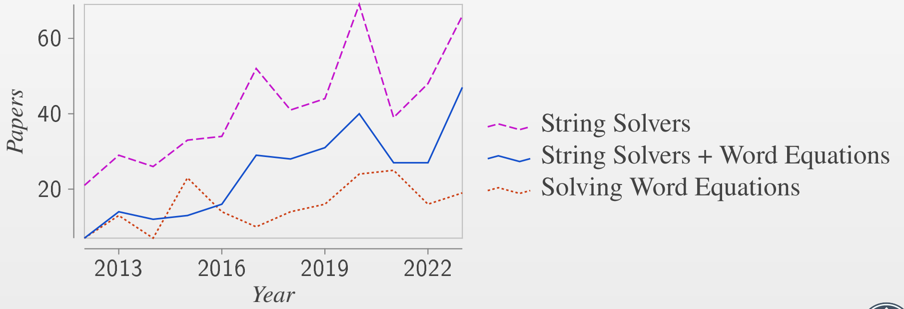
Simplified program

```
main'(X,Y,'i'++Z) = false;
main'(X,Y,'s'++Z) = false;
main'('a'++Y,Y,ε) = true;
main'(X,Y,ε)      = false;
```

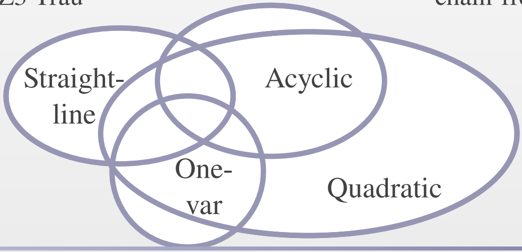
- Quadratic equation $WX = aWY$ ($W :=$ `g(Z)`) appears as a constraint generated by rule `eq(X, X) = true`, and its solution set includes $W \in a^*$, which is incompatible with any trace of g-computation except the trivial one.

In recent years:

- Many studies are conducted on developing automated reasoning tools capable of proving or disproving statements involving words.
  - Their task is to automatically determine the satisfiability of that formula.

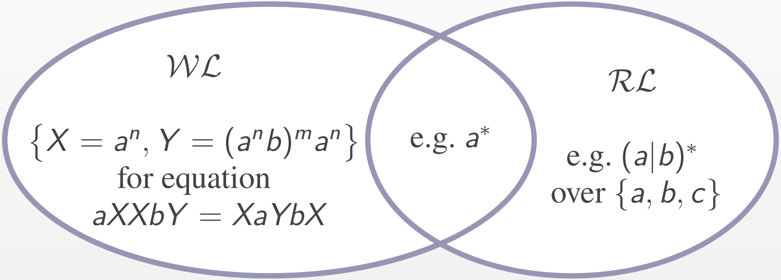


- Some string solving tools using word equations as constraint sets:
  - HAMPI — bounded
  - Norn — acyclic
  - OSTRICH — straight-line
  - Sloth — straight-line & acyclic
  - Woorpje — bounded
  - CertiStr — acyclic
  - MSCP-A — regularly-cyclic
  - CVC5, Z3Str3 — acyclic & bounded
  - Z3-Trau — chain-free



$\mathcal{WL}$ vs. $\mathcal{RL}$ (Expressivity)

The word equation language $\mathcal{WL}$ vs. the regular expression language $\mathcal{RL}$: intersect but are not subsets each of other:



— There is no word equation, which solution-set is represented by $(a|b)^*$ over the alphabet $\{a, b, c\}$.

- J. Karhumaki, F. Mignosi, W. Plandowski. The expressibility of languages and relations by word equations. Journal of the ACM (JACM), 47(3), pp: 483-505, (2000).

Unary Quantifier-Free Predicates

Given a predicate P expressed by a word equation E, let us assume that E depends on a single variable X.

- What string properties can be expressed by one-variable word equations?
- How the equations can be normalized, in order to represent language properties consistently?

Non-trivial question: *Can the solution-set be constructively described simpler as compared to the description provided by the original equation itself?*

Equations $aX = Xa$, $aaX = Xaa$, and $aXX = XXa$ all have the same solution set a^*.

Theorem (Nowotka-Saarela, 2018).
Every one-variable word equation has either infinitely many or at most three solutions.

Definition
A word $\eta \in \Sigma^+$ is said to be primitive, if for any $\xi \in \Sigma^*$, $n \in \mathbb{N}$ s.t. $\eta = \xi^n$ the equality $n = 1$ holds.

If an X-variable word equation has infinitely many solutions, then its whole solution-set is described with the fractional powers

$$\left\{ (\underbrace{\xi\zeta}_{\text{primitive word}})^n \xi \mid n \in \mathbb{N} \right\}$$

Lattice WL_0

We introduce a set of word-equation lattices over strings.

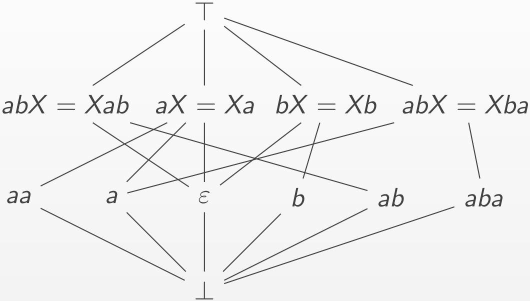
Lemma

One-variable equations with infinite solution sets and trivial equations of the form $X = \eta$, together with $\top$, $\perp$ form a complete lattice, that is:

- Given any constant words ξ_1, ξ_2, at most one infinite-solution one-variable word equation is satisfied by both.
- Given any two infinite-solution word equations $\xi_1\xi_2X = X\xi_2\xi_1$, $\xi_3\xi_4X = X\xi_4\xi_3$, at most one word in Σ^* satisfies both.

Base lattice WL_0 includes all elements $\{X = \xi\}$, $\{\xi_1\xi_2X = X\xi_2\xi_1\}$; its finite sublattices $\mathsf{WL}_0(\mathcal{P})$ capture properties of program $\mathcal{P}$.

A Lattice For $\{\varepsilon, a, a^2, b, ab, aba\}$ (Program Analysis*)



- A non-trivial one-variable word equation with an infinite solution set can be reduced to an equivalent one (where η_i stand for constant words)
$$\underbrace{\eta_1\eta_2 X = X\eta_2\eta_1 \ (\eta_1\eta_2 \text{ is primitive})}_{\text{normal-form condition}}.$$
- $\big((\eta_1\eta_2)^n X = X(\eta_2\eta_1)^n\big) \Leftrightarrow \big(\eta_1\eta_2 X = X\eta_2\eta_1\big)$ $(n \geq 1)$, hence the widening problem is resolved.

Lifting JS String Operations to Abstract Domain

Simple example — built-in string concatenation in JavaScript. We assume that x, y are non-bottom when they are concatenated, otherwise the concatenation results in an error.

$$x + y = \begin{cases} \{X = \xi_1\xi_2\}, \text{ if } x = \{X = \xi_1\}, y = \{X = \xi_2\}; \\ \big\{\xi_4\xi_5 X = X\xi_5\xi_4\big\}, \text{ if } x = \big\{X = \xi_1\big\} \;\&\; y = \big\{\xi_2\xi_3 X = X\xi_3\xi_2\big\} \\ \qquad \text{and } \xi_4, \xi_5 \text{ are s.t. } \xi_5\xi_4 = \xi_3\xi_2 \text{ and } \exists n\big(\xi_1\xi_2 = (\xi_4\xi_5)^n\xi_4\big); \\ \big\{\xi_4\xi_5 X = X\xi_5\xi_4\big\}, \text{ if } x = \big\{X = \xi_1\xi_2 X = X\xi_2\xi_1\big\} \;\&\; y = \big\{X = \xi_3\big\} \\ \qquad \text{and } \xi_4, \xi_5 \text{ are s.t. } \xi_4\xi_5 = \xi_1\xi_2 \text{ and } \exists n\big(\xi_1\xi_3 = (\xi_4\xi_5)^n\xi_4\big); \\ \big\{\xi_5\xi_6 X = X\xi_6\xi_5\big\}, \text{ if } x = \big\{\xi_1\xi_2 X = X\xi_2\xi_1\big\} \;\&\; y = \big\{\xi_3\xi_4 X = X\xi_4\xi_3\big\} \\ \qquad \text{and } \xi_2\xi_1 = \xi_3\xi_4 \\ \qquad \text{and } \xi_5, \xi_6 \text{ are s.t. } \xi_5\xi_6 = \xi_1\xi_2 \text{ and } \exists n\big(\xi_1\xi_3 = (\xi_1\xi_2)^n\xi_5\big); \\ \top, \text{ otherwise.} \end{cases}$$

Below we consider the only non-trivial case, when x and y are both infinite-solution equations.

Lifting JS String Operations to Abstract Domain

$$\boxed{\text{x + y}} = \begin{cases} \dots \\ \{\xi_5\xi_6 X = X\xi_6\xi_5\}, \text{ if } x = \{\xi_1\xi_2 X = X\xi_2\xi_1\} \text{ \& } y = \{\xi_3\xi_4 X = X\xi_4\xi_3\} \\ \hfill \text{and } \xi_2\xi_1 = \xi_3\xi_4 \\ \hfill \text{and } \xi_5, \xi_6 \text{ are s.t. } \xi_5\xi_6 = \xi_1\xi_2 \text{ and } \exists n\big(\xi_1\xi_3 = (\xi_1\xi_2)^n\xi_5\big); \\ \top, \text{ otherwise.} \end{cases}$$

Where do ξ_5, ξ_6 come from?

$$\exists \xi_5, \xi_6 \forall m, n \, \exists k \Big(\underbrace{(\xi_1\xi_2)^m\xi_1}_{x \text{ values}} \underbrace{(\xi_3\xi_4)^n\xi_3}_{y \text{ values}} = \underbrace{(\xi_5\xi_6)^k\xi_5}_{\boxed{\text{x+y}} \text{ values}} \Big)$$

$$\Downarrow$$

$$\big(\xi_1\xi_2 = \xi_5\xi_6\big) \text{ \& } \forall n \, \exists k \Big(\xi_1 \underbrace{(\xi_3\xi_4)^n\xi_3}_{y \text{ values}} = \underbrace{\xi_1 (\xi_2\xi_1)^k\xi_2}_{(\xi_5\xi_6)^{k+1}} \xi_5 \Big)$$

String Morphisms and Fixpoints

Classical Lemma

If σ is a solution to equation $\mathcal{U} = \mathcal{V}$, and h is a morphism, then $h \circ \sigma$ is a solution to $h(\mathcal{U}) = h(\mathcal{V})$.

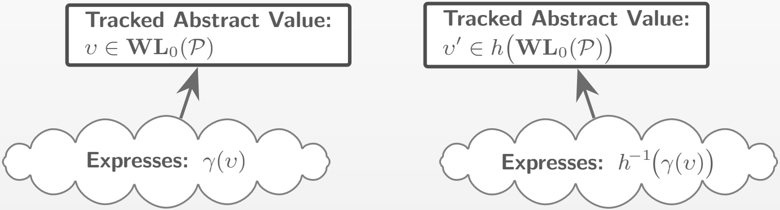
Corollary: string morphisms $h : \Sigma \to \Sigma^*$, extended to WL_0 elements by normalising the morphic images of the equation sides, are monotone lattice mappings. Namely,

$$\left(E_1 \preceq E_2\right) \Rightarrow \left(h(E_1) \preceq h(E_2)\right).$$

By Knaster–Tarski theorem, morphisms fixpoints form a complete sublattice of WL_0.

Expressibility via Transformations

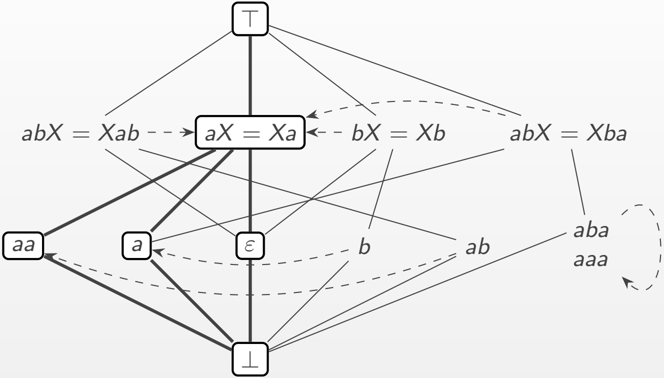
Given a monotone mapping h, we can track properties expressed by inverse morphisms of concretisations of the abstract values.



There $h^{-1} : 2^S \to 2^S$ is defined as usual:

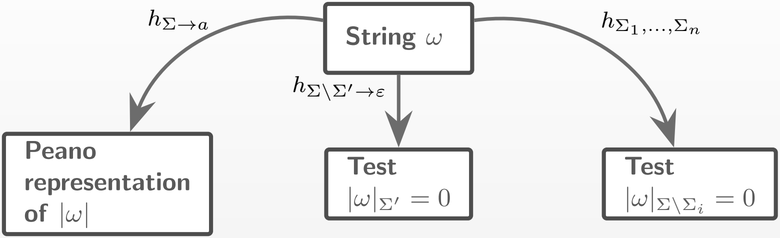
- $\forall s \in S\Big(h^{-1}\big(\{s\}\big) = \big\{s' \mid h(s') = s\big\}\Big)$;
- $h^{-1}(\mathcal{M}_1 \cup \mathcal{M}_2) = h^{-1}(\mathcal{M}_1) \cup h^{-1}(\mathcal{M}_2)$.

Fixpoint Example



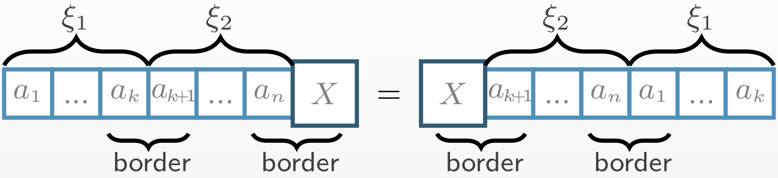
The sublattice generated by the morphism $h(x) = a$ tracks string lengths in a given program.

Expressibility of Fixpoint Lattices



- $h_{\Sigma\to a} \triangleq$ a morphism that maps a to itself and other letters to ε;
- $h_{\Sigma\setminus\Sigma'\to\varepsilon} \triangleq$ a morphism that maps all the letters from Σ' to a and all the other letters to ε.
- Let $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \dots \Sigma_n$, where $\forall i, j (i \neq j \Rightarrow \Sigma_i \cap \Sigma_j = \varnothing)$.
  $h_{\Sigma_1,\dots,\Sigma_k} \triangleq$ a morphism that maps all the elements of the disjoint sets to a single letter.

Inverse Images of String Morphisms



An inverse mapping of a morphism h is border-preserving wrt $\mathcal{L}$, if for any element $\{\xi_1\xi_2X = X\xi_2\xi_1\}$ of $\mathcal{L}$ and for any $a \in \Sigma$, $h(a)$ either contains no border of $\xi_1\xi_2X = X\xi_2\xi_1$, or is equal to ξ_2, and $\xi_1 = \varepsilon$.

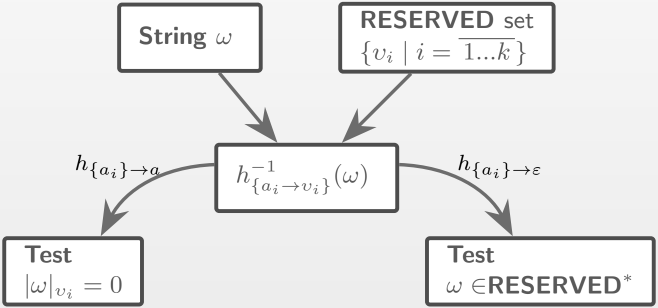
Lemma

Given lattice $\mathsf{WL}_0(\mathcal{P})$ and a string morphism h satisfying the conditions above, for any order induced on Σ, $h^{-1}_{\min}$ is a lattice morphism.

Expressibility of Inverse Mappings

Let RESERVED be a set of reserved words υ_i (keywords, constants, etc), s.t. morphism $h_{\{a_i \to \upsilon_i\}}$ mapping fresh letters from Σ' to υ_i is border-preserving.

Then an inverse image lattice $h^{-1}_{\{a_i \to \upsilon_i\}}$ can be used to track occurrences of the keywords inside strings.



Word Equations in Abstract Interpretation (Program Analysis)

Simple Example: abstract interpretation over $\mathsf{WL}_0(\mathcal{P})$-lattice shows the sanitization given in line 5 is sound (line 7 is unreachable).

```
1   z = ξ;
2   x = '<script' + z;
3   if (x != z + z) {
4      x = x + x };
5   replaceAll(x,'ip','ipv4');
6   if (contains(x,'script')) {
7      return 'Possible code injection'; }
```

Word Equations in Abstract Interpretation (Program Analysis)

The sanitization given in line 5 is sound.

1	`z = ξ;`	$z \mapsto \{X = \xi\}$
2	`x = '<script' + z;`	$x \mapsto \{X = \eta\}$
3	`if (x != z + z) {`	(does not change)
4	`x = x + x };`	$\{X = \eta\} \vee \{X = \eta + \eta\}$
		$= \{prm(\eta)X = X\,prm(\eta)\}$; $x \mapsto \{prm(\eta)X = X\,prm(\eta)\}$
5	`replaceAll(x,'ip','ipv4');`	$x \mapsto^{*} \{\eta' X = X \eta'\}$
6	`if (contains(x,'script')) {`	(never holds for the abstract value of x)
7	`return 'Possible code injection'; }`	(is pruned)

- $prm(\eta)$ is a primitive* root of η;
- $\eta = \texttt{'<script'} + \xi$;
  $\eta' = \texttt{replaceAll}(prm(\eta), \texttt{'ip'}, \texttt{'ipv4'})$.
- predicate `contains(x,'script')` fails for any value of X satisfying $\eta' X = X \eta'$.

Conclusion

- An infinite-solution subset of one-variable word equations together with predicates $\mathcal{L}att_{\mathrm{Eq}} = \{X = \xi\}$ form a finite-height lattice WL_0. Sublattices of the lattice WL_0 can be used for abstract interpretation of string-manipulating programs.

- Monotone lattice mappings by means of string morphisms and special cases of inverses of string morphisms generate fixpoint lattices upon WL_0, which can track additional program properties, non-expressible in the word equations language directly.

Work-In-Progress

- A word-equation-lattice-based framework for analysing real-world JavaScript string-manipulating programs.

- Lattice modifications capturing relations between values.

  A simple instance:
  $\left\{\mathsf{WL}_0(\mathcal{P})[X] \times \mathsf{WL}_0(\mathcal{P})[Y] \times \langle \top, \bot, \{XY = YX\}\rangle\right\}$.

Thanks for Attention

- Any questions?

Possible Extensions

$\mathcal{WL}$ — existential string theory

(concatenation + equality = word equations).

Theory	Replace All (Const Args)	letter count	length count	$\mathcal{RL}$	Complexity
$\mathcal{WL} + \mathcal{RL}$	✗	✗	✗	✓	PSPACE
$\mathcal{WL}$+len	✗	✗	✓	✗	???
$\mathcal{WL}$+count	✗	✓	✗	✗	Undec.
$\mathcal{WL}$+repl	✓	✗	✗	✗	Undec.

References-I

1. P.A. Abdulla, et al.: String Constraints for Verification. In: CAV 2014. LNCS, Vol. 8559, pp. 150–166 (2014).
2. P.A. Abdulla, et al.: Norn: an SMT solver for string constraints. In: CAV 2015. LNCS, vol. 9206, pp. 462–469 (2015).
3. P.A. Abdulla, et al.: Flatten and conquer: a framework for efficient analysis of string constraints, PLDI, 2017, ACM, pp. 602–617.
4. H. Barbosa, et al.: CVC5: a versatile and industrial-strength SMT solver. In: TACAS 2022. LNCS, vol. 13243, pp. 415–442, (2022).
5. J. Berstel, and J. Karhumaki. Combinatorics on words: a tutorial. Bulletin of the EATCS 79.178 (2003): 9.
6. E. Czeizler, The non-parametrizability of the word equation xyz = zvx: a short proof. Theoret. Comput. Sci. 345(2–3), 296–303 (2005).

. . .

References-II

7. T. Chen, M. Hague, A.W. Lin, P. Rummer, Z. Wu: Decision procedures for path feasibility of string-manipulating programs with complex operations. Proc. ACM Program. Lang. 3(POPL), 1–30 (2019).
8. J.D. Day, et al.: On solving word equations using SAT. In: RP 2019. LNCS, vol. 11674, pp. 93–106 (2019).
9. J.D. Day. Word Equations in the Context of String Solving. In Int. Conf. on Developments in Language Theory. 2022, pp. 13-32.
10. L. Holik, P. Janku, A.W. Lin, P. Rummer, T. Vojnar: String constraints with concatenation and transducers solved efficiently. In: Proc. of the ACM on Programming Languages, vol. 2, pp. 1–32 (2018).
11. L. Ilie, W. Plandowski. Two-variable word equations. RAIRO-Theoret. Inform. Appl. 34(6), 467–501 (2000).

. . .

References-III

12. A. Jez. Recompression: a simple and powerful technique for word equations. J. ACM (JACM) 63(1), 1–51 (2016).
13. S. Kan, A.W. Lin, P. Rummer, M. Schrader: CertiStr: a certified string solver. In: Proc. of the 11th ACM SIGPLAN International Conf. on Certified Programs and Proofs, pp. 210–224 (2022).
14. J. Karhumaki, F. Mignosi, and W. Plandowski, The expressibility of languages and relations by word equations, J. ACM, vol. 47, no. 3, pp. 483–505, May 2000.
15. A. Kiezun, V. Ganesh, S. Artzi, P.J. Guo, P. Hooimeijer, M.D. Ernst: HAMPI: a solver for word equations over strings, regular expressions, and context-free grammars. ACM Trans. Softw. Eng. Methodol. (TOSEM) 21(4), 1–28 (2013).

. . .

References-IV

16. Yu. I. Khmelevskii, Equations in a free semigroup, Trudy Mat. Inst. Steklov., 107, 1971, 3–288; Proc. Steklov Inst. Math., 107 (1971), pp: 1–270.
17. N.K. Kosovskij, Some properties of solutions to equations in a semigroup (in Russian), Zap. Nauch. Sem. LOMI, vol. 32, pp. 21–28, 1972. Available at `https://www.mathnet.ru/rus/znsl2560`
18. G.S. Makanin, The problem of solvability of equations in a free semigroup, Math. USSR-Sb., 32:2 (1977), pp: 129–198.
19. F. Mora, M. Berzish, M. Kulczynski, D. Nowotka, V. Ganesh: Z3Str4: a multi-armed string solver. In: FM 2021. LNCS, vol. 13047, pp. 389–406, (2021).

References-V

20. D. Nowotka, and A. Saarela. An optimal bound on the solution sets of one-variable word equations and its consequences. SIAM Journal on Computing 51.1 (2022): pp: 1-18.
21. W. Plandowski. An efficient algorithm for solving word equations. In: Proc. of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, pp. 467–476 (2006).
22. W. Plandowski. An efficient algorithm for solving word equations. Theoretical Computer Science Volume 792, 5 November 2019, Pages 20-61.
23. A.A. Razborov. On systems of equations in free groups. In: Combinatorial and Geometric Group Theory, pp. 269–283 (1993).