A Uniform Framework for Handling Position Constraints in String Solving

Yu-Fang Chen[1] Vojtěch Havlena[2] Michal Hečko[2]
Lukáš Holík[2] **Ondřej Lengál**[2]

[1]Academia Sinica, Taiwan
[2]Brno University of Technology, Czech Republic

PLDI'25

String Solving

- Satisfiability of formulas over string constraints such as:

$$\underbrace{x = yz}_{\text{equations}} \land \overbrace{yz \neq ua}^{\text{disequalities}} \land \underbrace{x \in (ab)^* a^+ (b|c)}_{\text{regular constraints}} \land \overbrace{|xy| = 2|uv| + 1}^{\text{length constraints}} \land \underbrace{\neg contains(uxz, zbcx)}_{\text{more complex operations}}$$

String Solving

- Satisfiability of formulas over string constraints such as:

$$\underbrace{x = yz}_{\text{equations}} \wedge \overbrace{yz \neq ua}^{\text{disequalities}} \wedge \underbrace{x \in (ab)^* a^+ (b|c)}_{\text{regular constraints}} \wedge \overbrace{|xy| = 2|uv| + 1}^{\text{length constraints}} \wedge \underbrace{\neg contains(uxz, zbcx)}_{\text{more complex operations}}$$

- Reasoning about string manipulation in programs
  - source of security vulnerabilities (SQL/code injection, cross-site scripting)
  - scripting languages rely heavily on strings
- Analysis of AWS/Rego access policies
- ...
- implemented in solvers:
  - Z3, CVC5, ... (deduction-based)
  - NORN, TRAU, SLOTH, OSTRICH, Z3-NOODLER, ... (automata-based)

String Solving

- Satisfiability of formulas over string constraints such as:

$$\underbrace{x = yz}_{\text{equations}} \wedge \overbrace{yz \neq ua}^{\text{disequalities}} \wedge \underbrace{x \in (ab)^* a^+ (b|c)}_{\text{regular constraints}} \wedge \overbrace{|xy| = 2|uv| + 1}^{\text{length constraints}} \wedge \underbrace{\neg contains(uxz, zbcx)}_{\text{more complex operations}}$$

- Reasoning about string manipulation in programs
  - source of security vulnerabilities (SQL/code injection, cross-site scripting)
  - scripting languages rely heavily on strings
- Analysis of AWS/Rego access policies
- ...
- implemented in solvers:
  - Z3, CVC5, ... (deduction-based)
  - NORN, TRAU, SLOTH, OSTRICH, Z3-NOODLER, ... (automata-based)

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$
- disequalities: $xyz \neq uawx$

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$
- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$
- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$
- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$
- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$
- not contains: $\neg contains(xya, zyw)$

The Fragment

Here, we consider:

- (monadic) regular constraints: $x \in (ab)^* a^+ (b|c)$
- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$
- not contains: $\neg contains(xya, zyw)$

(the last five items together:) position constraints

Position Constraints

Position Constraints

- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$
- not contains: $\neg contains(xya, zyw)$

} position constraints

Why "position constraints"?

Position Constraints

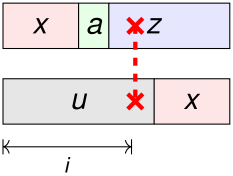
- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$
- not contains: $\neg contains(xya, zyw)$

} position constraints

Why “position constraints”?

- they are satisfied by “existence of an interesting position”
- e.g.,

$$xaz \neq ux \quad \Leftrightarrow \quad \exists i \in \mathbb{N} \colon (xaz)[i] \neq (ux)[i]$$



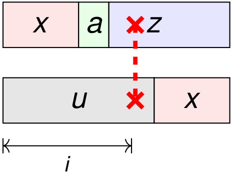
Position Constraints

- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$
- not contains: $\neg contains(xya, zyw)$

} position constraints

Why "position constraints"?

- they are satisfied by "existence of an interesting position"
- e.g.,



$$xaz \neq ux \quad \Leftrightarrow \quad \exists i \in \mathbb{N}\colon (xaz)[i] \neq (ux)[i]$$

$$\neg suffixof(axb, yzw) \quad \Leftrightarrow \quad \exists i \in \mathbb{N}\colon i < |axb| \wedge (axb)[-i] \neq (yzw)[-i]$$

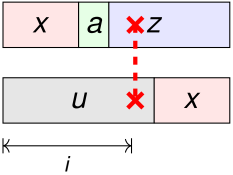
Position Constraints

- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$
- not prefix, not suffix: $\neg suffixof(axb, yzw)$
- not contains: $\neg contains(xya, zyw)$

} position constraints

Why "position constraints"?

- they are satisfied by "existence of an interesting position"
- e.g.,



$$xaz \neq ux \quad \Leftrightarrow \quad \exists i \in \mathbb{N} \colon (xaz)[i] \neq (ux)[i]$$

$$\neg suffixof(axb, yzw) \quad \Leftrightarrow \quad \exists i \in \mathbb{N} \colon i < |axb| \land (axb)[-i] \neq (yzw)[-i]$$

$$\neg contains(xya, zyw) \quad \Leftrightarrow \quad \forall k \colon 0 \leq k \leq |zyw| - |xya| \Rightarrow \exists i \colon (xya)[i] \neq (zyw)[i + k]$$

Our Approach (High Level)

1. Construct the tag automaton $\mathcal{A}_{tag}$ encoding positions' information
   - a version of nondeterministic finite automaton with additional information on edges

Our Approach (High Level)

1. Construct the tag automaton $\mathcal{A}_{tag}$ encoding positions' information
   - a version of nondeterministic finite automaton with additional information on edges

2. Construct the Parikh formula $PF(\mathcal{A}_{tag})$
   - a linear integer arithmetic (LIA) formula encoding information about the runs of $\mathcal{A}_{tag}$

Theorem (Parikh's theorem (modified))

Numbers of occurrences of symbols in words in a regular language can be described by a linear integer arithmetic formula.

Our Approach (High Level)

1. Construct the tag automaton $\mathcal{A}_{tag}$ encoding positions' information
   - a version of nondeterministic finite automaton with additional information on edges
2. Construct the Parikh formula $PF(\mathcal{A}_{tag})$
   - a linear integer arithmetic (LIA) formula encoding information about the runs of $\mathcal{A}_{tag}$

   Theorem (Parikh's theorem (modified))

   Numbers of occurrences of symbols in words in a regular language can be described by a linear integer arithmetic formula.

3. Solve $PF(\mathcal{A}_{tag})$ by an off-the-shelf LIA solver

Tag Automaton

Tag automaton over set of tags $\mathbb{T}$:

- extension of finite automaton
- $\mathcal{A}_{tag} = (Q, \Delta, I, F)$
  - Q: (finite) set of states
  - $I \subseteq Q$: initial states
  - $F \subseteq Q$: final states
  - $\Delta \subseteq Q \times 2^{\mathbb{T}} \times Q$: transitions

Tag Automaton

Tag automaton over set of tags $\mathbb{T}$:

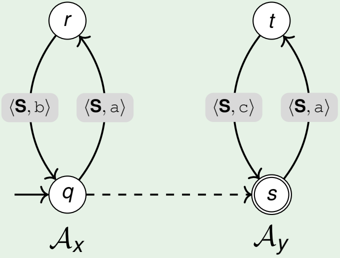
- extension of finite automaton
- $\mathcal{A}_{tag} = (Q, \Delta, I, F)$
  - Q: (finite) set of states
  - $I \subseteq Q$: initial states
  - $F \subseteq Q$: final states
  - $\Delta \subseteq Q \times 2^{\mathbb{T}} \times Q$: transitions

Example

$x \in (ab)^*$

$y \in (ac)^*$

$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\}$



Tag Automaton

Tag automaton over set of **tags** $\mathbb{T}$:

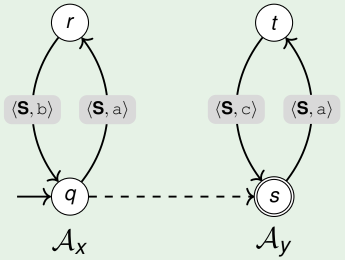
- extension of **finite automaton**
- $\mathcal{A}_{tag} = (Q, \Delta, I, F)$
  - Q: (finite) set of states
  - $I \subseteq Q$: initial states
  - $F \subseteq Q$: final states
  - $\Delta \subseteq Q \times 2^{\mathbb{T}} \times Q$: transitions

Example

$x \in (ab)^*$

$y \in (ac)^*$

$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\}$



Parikh formula $PF(\mathcal{A}_{tag})$:

- a **linear integer arithmetic** (LIA) formula over variables $\#\mathbb{T} = \{\#t \mid t \in \mathbb{T}\}$
- assignments $\{\#t \mapsto n_t \mid t \in \mathbb{T}, n_t \in \mathbb{N}\}$ (simplified)
- $m \models PF(\mathcal{A}_{tag})$ iff there is an accepting run in $\mathcal{A}_{tag}$ s.t. $m(\#t)$ is the number of occurrences of a tag in a word accepted by $\mathcal{A}_{tag}$.
- e.g., if $ababacac \in L(\mathcal{A}_{tag})$ then $\{\#\langle \mathbf{S}, a\rangle = 4, \#\langle \mathbf{S}, b\rangle = 2, \#\langle \mathbf{S}, c\rangle = 2\} \models PF(\mathcal{A}_{tag})$

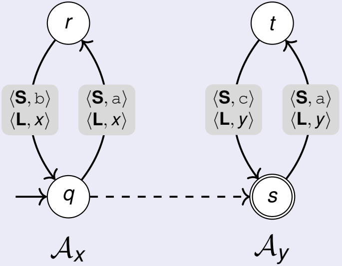
Length Constraints: $x \in (ab)^* \land y \in (ac)^* \quad \land \quad 2|x| = 3|y| + 2$

Length Constraints: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad 2|x| = 3|y| + 2$

- construct a tag automaton by connecting the NFAs for x and y
- tags

$$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\}$$

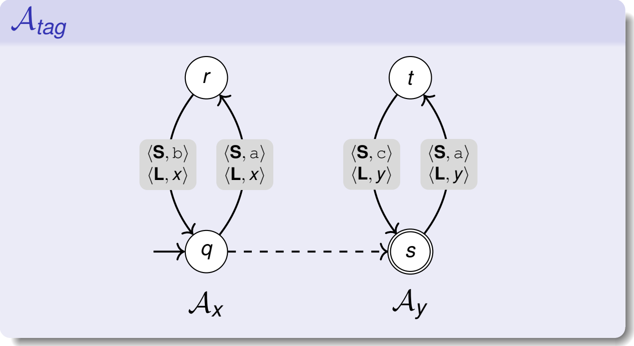
$\mathcal{A}_{tag}$



Length Constraints: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad 2|x| = 3|y| + 2$

- construct a tag automaton by connecting the NFAs for x and y
- tags

$$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\}$$



$$\varphi\colon PF(\mathcal{A}_{tag}) \quad \wedge \quad 2 \cdot \#\langle \mathbf{L}, x\rangle = 3 \cdot \#\langle \mathbf{L}, y\rangle + 2$$

Single Simple Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad x \neq y$

Single Simple Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad x \neq y$

- construct tag automaton $\mathcal{A}_{tag}$
- tags:

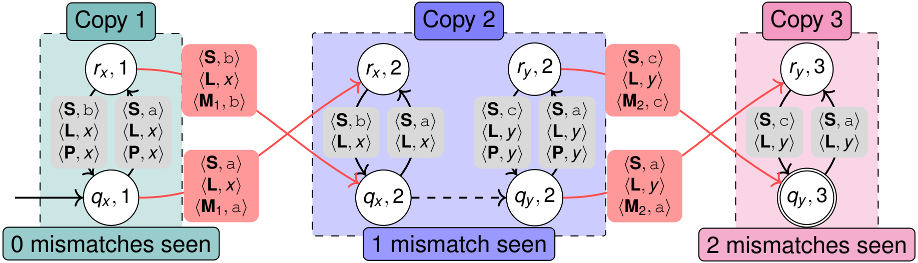
$$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\} \cup \{\langle \mathbf{P}, x\rangle, \langle \mathbf{P}, y\rangle\} \cup \{\langle \mathbf{M}_1, a\rangle, \langle \mathbf{M}_1, b\rangle, \langle \mathbf{M}_2, a\rangle, \langle \mathbf{M}_2, c\rangle\}$$

Single Simple Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad x \neq y$

- construct tag automaton $\mathcal{A}_{tag}$
- tags:

$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\} \cup \{\langle \mathbf{P}, x\rangle, \langle \mathbf{P}, y\rangle\} \cup \{\langle \mathbf{M}_1, a\rangle, \langle \mathbf{M}_1, b\rangle, \langle \mathbf{M}_2, a\rangle, \langle \mathbf{M}_2, c\rangle\}$

- count the positions before the mismatch in x ($\#\langle \mathbf{P}, x\rangle$)
- count the positions before the mismatch in y ($\#\langle \mathbf{P}, y\rangle$)
- check that $\#\langle \mathbf{P}, x\rangle = \#\langle \mathbf{P}, y\rangle$ and there is a mismatch

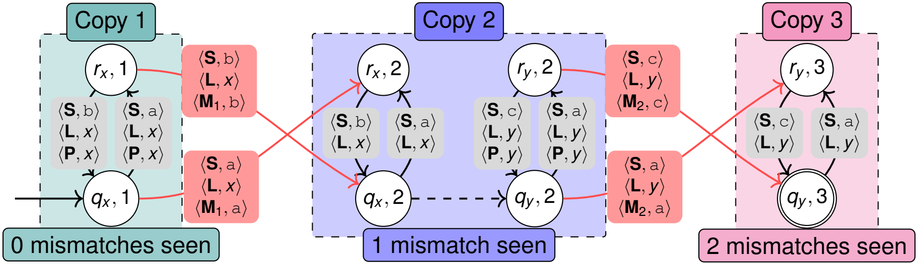


Single Simple Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad x \neq y$

- construct tag automaton $\mathcal{A}_{tag}$
- tags:

$$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\} \cup \{\langle \mathbf{P}, x\rangle, \langle \mathbf{P}, y\rangle\} \cup \{\langle \mathbf{M}_1, a\rangle, \langle \mathbf{M}_1, b\rangle, \langle \mathbf{M}_2, a\rangle, \langle \mathbf{M}_2, c\rangle\}$$

- count the positions before the mismatch in x ($\#\langle \mathbf{P}, x\rangle$)
- count the positions before the mismatch in y ($\#\langle \mathbf{P}, y\rangle$)
- check that $\#\langle \mathbf{P}, x\rangle = \#\langle \mathbf{P}, y\rangle$ and there is a mismatch

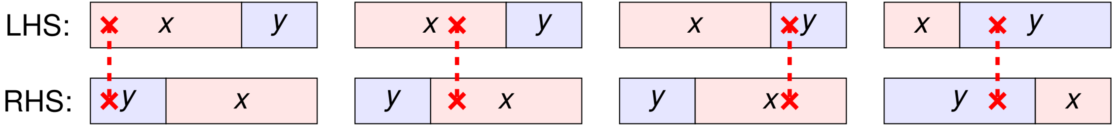


$$\varphi\colon PF(\mathcal{A}_{tag}) \wedge \#\langle \mathbf{P}, x\rangle = \#\langle \mathbf{P}, y\rangle \wedge \bigwedge_{a \in \Sigma} (\#\langle \mathbf{M}_1, a\rangle + \#\langle \mathbf{M}_2, a\rangle < 2)$$

Single (not simple) Disequality: $x \in (ab)^* \land y \in (ac)^* \quad \land \quad xy \neq yx$

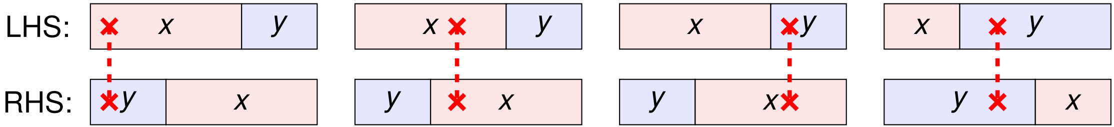
Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$

- now we need to consider several options:



Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$

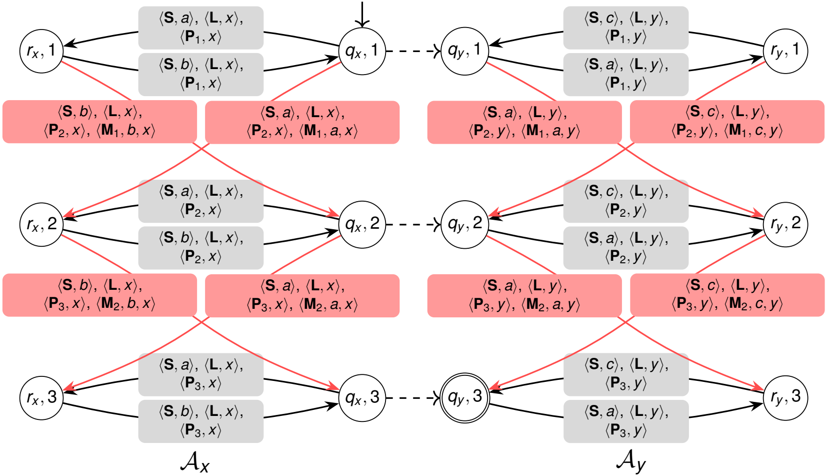
- now we need to consider several options:



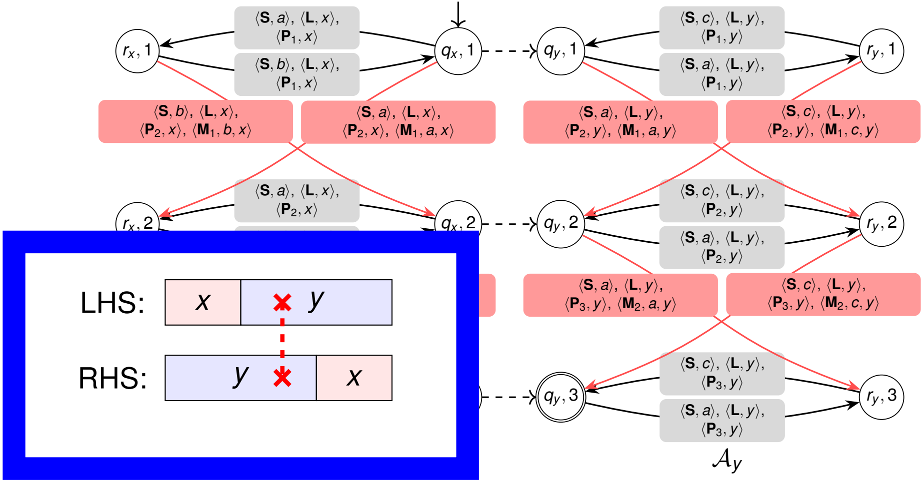
- construct tag automation $\mathcal{A}_{tag}$
- tags:

$$
\begin{aligned}
\mathbb{T} = \; & \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \\
& \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\} \cup \\
& \{\langle \mathbf{P}_1, x\rangle, \langle \mathbf{P}_1, y\rangle, \langle \mathbf{P}_2, x\rangle, \langle \mathbf{P}_2, y\rangle\} \cup \\
& \{\langle \mathbf{M}_1, a, x\rangle, \langle \mathbf{M}_1, b, x\rangle, \langle \mathbf{M}_1, a, y\rangle, \langle \mathbf{M}_1, c, y\rangle, \\
& \;\langle \mathbf{M}_2, a, x\rangle, \langle \mathbf{M}_2, b, x\rangle, \langle \mathbf{M}_2, a, y\rangle, \langle \mathbf{M}_2, c, y\rangle\}
\end{aligned}
$$

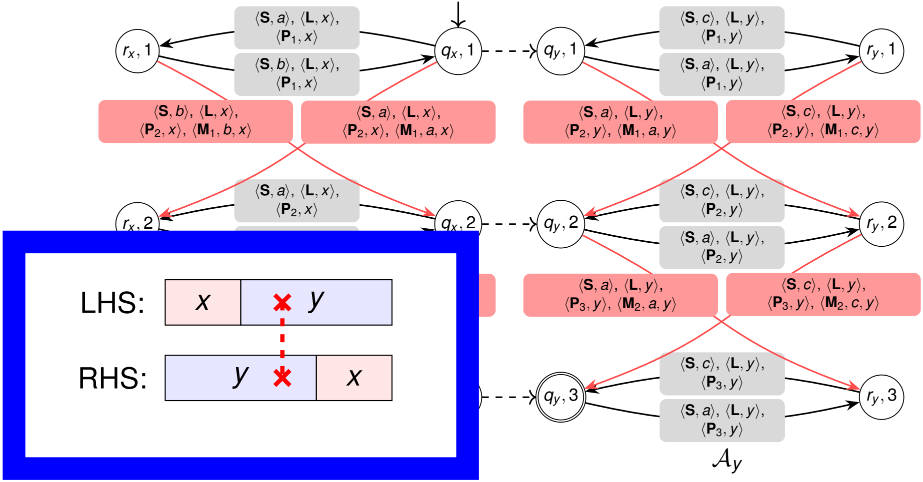
Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$



Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$



Single (not simple) Disequality: $x \in (ab)^* \land y \in (ac)^* \quad \land \quad xy \neq yx$



$$\varphi\colon PF(\mathcal{A}_{tag}) \land (\#\langle \mathbf{L}, x \rangle + \#\langle \mathbf{P}_1, y \rangle = \#\langle \mathbf{P}_1, y \rangle + \#\langle \mathbf{P}_2, y \rangle) \land$$
$$(\#\langle \mathbf{M}_1, y, a \rangle + \#\langle \mathbf{M}_2, y, a \rangle < 2) \land (\#\langle \mathbf{M}_1, y, c \rangle + \#\langle \mathbf{M}_2, y, c \rangle < 2) \land \ldots$$

Multiple Disequalities: $x \neq y \wedge x \neq z$

Multiple Disequalities: $x \neq y \land x \neq z$

- we need to consider a mismatch for $x \neq y$ and a mismatch for $x \neq z$
- $\rightsquigarrow$ more copies
- naive solution: $\frac{(2n)!}{2^n} \in 2^{\Theta(n \log n)}$
  - $n \ldots$ number of disequalities

Multiple Disequalities: $x \neq y \land x \neq z$

- we need to consider a mismatch for $x \neq y$ and a mismatch for $x \neq z$
- $\rightsquigarrow$ more copies
- naive solution: $\frac{(2n)!}{2^n} \in 2^{\Theta(n \log n)}$
  - $n \ldots$ number of disequalities
- better encoding: $\mathcal{O}(n)$
- details in the paper!

Multiple Disequalities: $x \neq y \wedge x \neq z$

- we need to consider a mismatch for $x \neq y$ and a mismatch for $x \neq z$
- $\rightsquigarrow$ more copies
- naive solution: $\frac{(2n)!}{2^n} \in 2^{\Theta(n \log n)}$
  - $n \ldots$ number of disequalities
- better encoding: $\mathcal{O}(n)$
- details in the paper!

Other constraints

- $\neg \mathit{prefixof}$, $\neg \mathit{suffixof}$, $\mathit{str.at}$, $\neg \mathit{str.at}$: similar technique
- $\neg \mathit{contains}$: $\ldots$

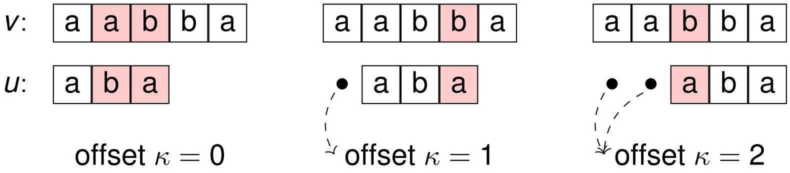
Not Contains: $\neg contains(u, v) \land u \in \ldots \land v \in \ldots$

Not Contains: $\neg contains(u, v) \land u \in \ldots \land v \in \ldots$

- implicit universal quantification

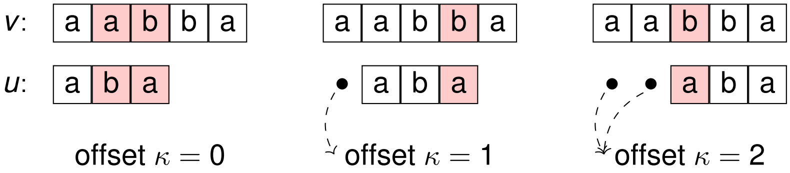
Not Contains: $\neg contains(u, v) \wedge u \in \ldots \wedge v \in \ldots$

- implicit universal quantification



Not Contains: $\neg contains(u, v) \land u \in \ldots \land v \in \ldots$

- implicit universal quantification



$$\forall \kappa \geq 0 \colon \exists \#\delta_1, \ldots \#\delta_n \colon PF(\mathcal{A}_{tag}) \land \ldots$$

Not Contains: $\neg contains(u, v) \land u \in \ldots \land v \in \ldots$

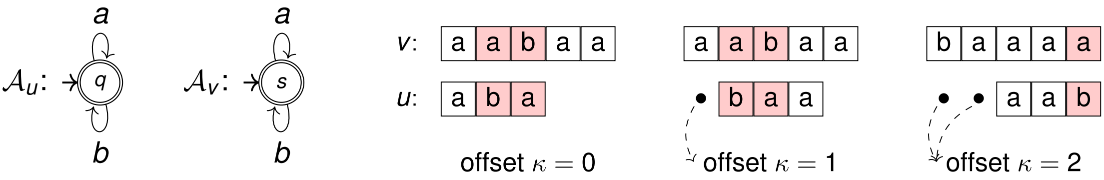
$$\forall \kappa \geq 0 \colon \exists \#\delta_1, \ldots \#\delta_n \colon PF(\mathcal{A}_{tag}) \land \ldots$$

- universal quantification of the offset κ doesn't work!
- one model of $PF(\mathcal{A}_{tag})$ may correspond to several different words
- $\rightsquigarrow$ allows different models for different κ!

Not Contains: $\neg contains(u, v) \land u \in \ldots \land v \in \ldots$

$$\forall \kappa \geq 0 \colon \exists \#\delta_1, \ldots \#\delta_n \colon PF(\mathcal{A}_{tag}) \land \ldots$$

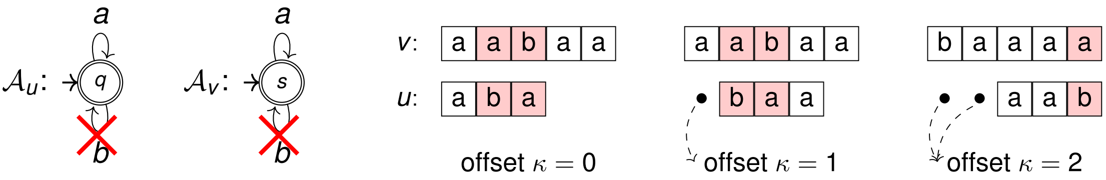
- universal quantification of the offset κ doesn't work!
- one model of $PF(\mathcal{A}_{tag})$ may correspond to several different words
- $\rightsquigarrow$ allows different models for different κ!
- example: suppose $m = \{\#a \mapsto 4, \#b \mapsto 1\}$



Not Contains: $\neg contains(u, v) \wedge u \in \ldots \wedge v \in \ldots$

$$\forall \kappa \geq 0 \colon \exists \#\delta_1, \ldots \#\delta_n \colon PF(\mathcal{A}_{tag}) \wedge \ldots$$

- universal quantification of the offset κ doesn't work!
- one model of $PF(\mathcal{A}_{tag})$ may correspond to several different words
- $\rightsquigarrow$ allows different models for different κ!
- example: suppose $m = \{\#a \mapsto 4, \#b \mapsto 1\}$



- restriction to flat regular constraints
  - $\forall\exists$ LIA formula
- details in the paper!

Decidability and Complexity

REG: regular language membership constraints $\mathcal{R}$

Decidability and Complexity

REG: regular language membership constraints $\mathcal{R}$

- REG with single disequality $x_1 \ldots x_m \neq y_1 \ldots y_n$:
  - PTIME ... $\mathcal{O}(nm \cdot |\Sigma|^3 \cdot |\mathcal{R}|^6)$
  - also for $\neg$*prefixof*, $\neg$*suffixof*

Decidability and Complexity

REG: regular language membership constraints $\mathcal{R}$

- REG with single disequality $x_1 \ldots x_m \neq y_1 \ldots y_n$:
  - PTIME ... $\mathcal{O}(nm \cdot |\Sigma|^3 \cdot |\mathcal{R}|^6)$
  - also for $\neg$*prefixof*, $\neg$*suffixof*
- REG with multiple disequalities $\bigwedge_{q \leq i \leq K}(x_{i,1} \ldots x_{i,m_i} \neq y_{i,1} \ldots y_{i,n_i})$
  - NP-complete
  - also for $\neg$*prefixof*, $\neg$*suffixof*, *str.at*, $\neg$*str.at*, lengths

Decidability and Complexity

REG: regular language membership constraints $\mathcal{R}$

- REG with single disequality $x_1 \dots x_m \neq y_1 \dots y_n$:
  - PTIME ... $\mathcal{O}(nm \cdot |\Sigma|^3 \cdot |\mathcal{R}|^6)$
  - also for $\neg$*prefixof*, $\neg$*suffixof*
- REG with multiple disequalities $\bigwedge_{q \leq i \leq K}(x_{i,1} \dots x_{i,m_i} \neq y_{i,1} \dots y_{i,n_i})$
  - NP-complete
  - also for $\neg$*prefixof*, $\neg$*suffixof*, *str.at*, $\neg$*str.at*, lengths
  - adding flat $\neg$*contains*:
    - NP-HARD (just one flat $\neg$*contains* suffices)
    - in NEXPTIME

Decidability and Complexity

REG: regular language membership constraints $\mathcal{R}$

- REG with single disequality $x_1 \ldots x_m \neq y_1 \ldots y_n$:
  - PTIME $\ldots \mathcal{O}(nm \cdot |\Sigma|^3 \cdot |\mathcal{R}|^6)$
  - also for $\neg$*prefixof*, $\neg$*suffixof*
- REG with multiple disequalities $\bigwedge_{q \leq i \leq K}(x_{i,1} \ldots x_{i,m_i} \neq y_{i,1} \ldots y_{i,n_i})$
  - NP-complete
  - also for $\neg$*prefixof*, $\neg$*suffixof*, *str.at*, $\neg$*str.at*, lengths
  - adding flat $\neg$*contains*:
    - NP-HARD (just one flat $\neg$*contains* suffices)
    - in NEXPTIME
- REG with multiple position constraints, lengths, and chain-free word equations
  - decidable (in ELEMENTARY)
  - efficient in practice!

Experimental Evaluation

- implemented in **Z3-NOODLER-POS** — extension of Z3-NOODLER
- compared to
  - Z3-NOODLER
  - CVC5
  - Z3
  - OSTRICH
- benchmarks:
  - **symbolic execution**[1]: using Python PyCT symbolic executor
    - biopython (77,222): bioinformatics Python tools
    - django (52,643): Django Python web app
    - thefuck (19,872): Python command mistake correction tool
  - **hand-crafted**:
    - position-hard (550): difficult small formulae with $\neq$ and $\neg$*contains*

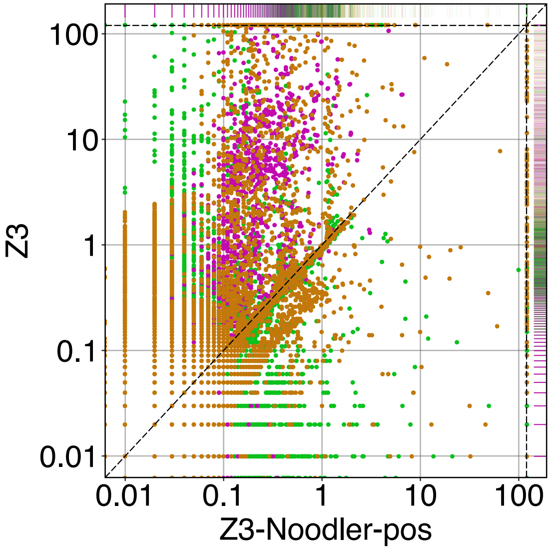
[1] Abdulla et al. "Solving not-substring constraint with flat abstraction". In: *APLAS'21*.

Experimental Evaluation

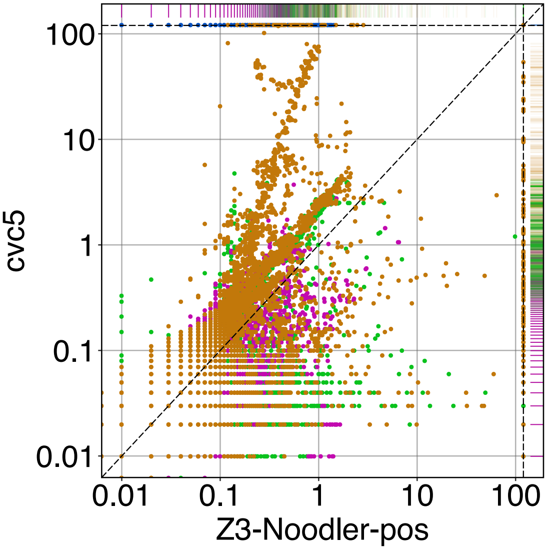
	biopython (77,222)		django (52,643)		thefuck (19,872)		position-hard (550)		All (150,287)	
	Unsolved	TimeAll	Unsolved	TimeAll	Unsolved	TimeAll	Unsolved	TimeAll	Unsolved	TimeAll
Z3-NOODLER-POS	171	24,010	39	8,005	0	**665**	0	**124**	210	**32,804**
Z3-NOODLER	507	64,385	145	20,873	376	45,757	480	59,512	1,508	190,527
CVC5	69	**21,114**	0	**4,515**	0	690	550	66,000	619	92,319
Z3	1,047	141,301	502	67,741	47	15,097	550	66,000	2,146	290,139
OSTRICH	2,986	1,108,306	4,404	1,507,806	967	236,192	550	66,000	8,907	2,918,304

- **Unsolved**: out of resources (timeout: 120 s) or Unk
- **TimeAll**: time-of-solved + (timeout $*$ #-of-failed-instances)

Comparison with Z3 and cvc5



(a) vs. Z3



(b) vs. cvc5

Conclusion and Future Work

Takeaway:

- regular + position constraints $\rightsquigarrow$
  - $\rightsquigarrow$ tag automaton $\rightsquigarrow$
  - $\rightsquigarrow$ Parikh formula $\rightsquigarrow$
  - $\rightsquigarrow$ LIA solver
- efficient in practice!

Conclusion and Future Work

Takeaway:

- regular + position constraints $\rightsquigarrow$
  - $\rightsquigarrow$ tag automaton $\rightsquigarrow$
  - $\rightsquigarrow$ Parikh formula $\rightsquigarrow$
  - $\rightsquigarrow$ LIA solver
- efficient in practice!

Future Work:

- non-flat $\neg$*contains*
  - REG + 1 non-flat not-contains: decidable in EXPSPACE (under review)
- extend to richer REG constraints
  - backreferences, bounded repetition, ...

Conclusion and Future Work

Takeaway:

- regular + position constraints $\rightsquigarrow$
  - $\rightsquigarrow$ tag automaton $\rightsquigarrow$
  - $\rightsquigarrow$ Parikh formula $\rightsquigarrow$
  - $\rightsquigarrow$ LIA solver
- efficient in practice!

Future Work:

- non-flat $\neg$*contains*
  - REG + 1 non-flat not-contains: decidable in EXPSPACE (under review)
- extend to richer REG constraints
  - backreferences, bounded repetition, ...

Thank you!

Word Equations: $uxa = buw$

Word Equations: $uxa = buw$

- word equations
  - $\rightsquigarrow$ can be transformed to regular constraints
    - basic algorithm of main automata-based solvers (NORN, OSTRICH, Z3-NOODLER, ...)
    - obtain the so-called monadic decomposition
    - incomplete, but mostly works in practice
- non-negated predicates: can be encoded as word equations
  - e.g., $\mathit{prefixof}(x, y) \quad \Leftrightarrow \quad x = yz$

Standard Approach for Position Constraints

Standard approach for handling position constraints:

- encode to word equations
- disequalities: $x \neq y$:

$$\bigvee_{\substack{c_1, c_2 \in \Sigma \\ c_1 \neq c_2}} \alpha c_1 \beta = xyz \land \alpha c_2 \beta' = uawx$$

- other constraints: similar
- solving word equations is in PSPACE (but often solved by algorithms)
- breaks chain-freeness

Standard Approach for Position Constraints

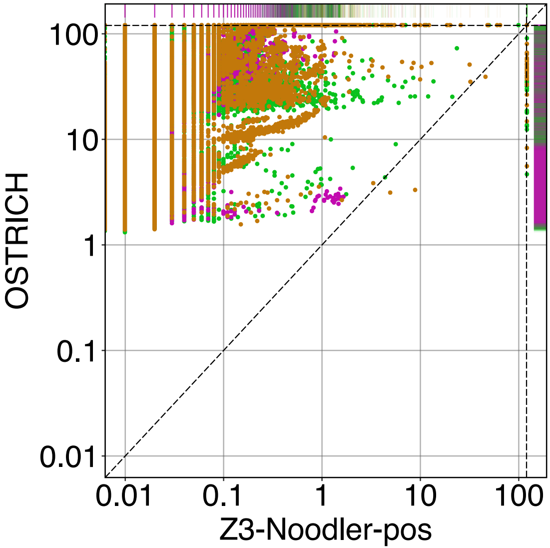
Standard approach for handling position constraints:

- encode to word equations
- disequalities: $x \neq y$:

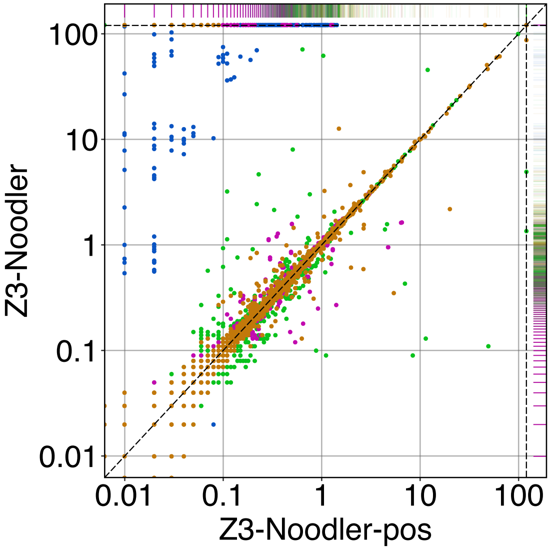
$$\bigvee_{\substack{c_1, c_2 \in \Sigma \\ c_1 \neq c_2}} \alpha c_1 \beta = xyz \wedge \alpha c_2 \beta' = uawx$$

- other constraints: similar
- solving word equations is in PSPACE (but often solved by algorithms)
- breaks chain-freeness
- $\neg$*contains*: no standard way
  - can be encoded in the $\forall\exists$ fragment of string constraints (undecidable)
  - $\neg contains(x, y)$

Comparison with OSTRICH and Z3-NOODLER



(a) Z3-NOODLER-POS vs. OSTRICH



(b) Z3-NOODLER-POS vs. Z3-NOODLER