Handling Position Constraints Uniformly Including the Negated Containment

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

MOSCA'25

Talk organization

Talk is composed of two parts:

1. Uniform framework for position constraints
2. Decidability of the negated containment predicate

Part 1: Uniform framework for position constraints

(Published at PLDI'25)

High-level: What kinds of formulae we target

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

We target (conjunctions of) *position constraints*:

- disequalities: $xyz \neq uawx$

High-level: What kinds of formulae we target

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

We target (conjunctions of) *position constraints*:

- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$

High-level: What kinds of formulae we target

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

We target (conjunctions of) *position constraints*:

- disequalities: $xyz \neq uawx$
- length constraints: $|xy| \leq 2|uaw| - 3$
- symbol (not) at position: $a = str.at(xy, 42)$

High-level: What kinds of formulae we target

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

We target (conjunctions of) *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: $\neg suffixof(axb, yzw)$

High-level: What kinds of formulae we target

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

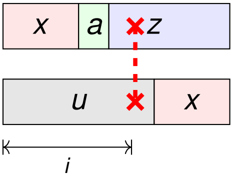
We target (conjunctions of) *position constraints*:

- 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: $\neg contains(xya, zyw)$

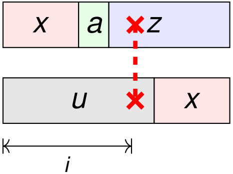
Why "Position Constraints"?

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

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



Why "Position Constraints"?

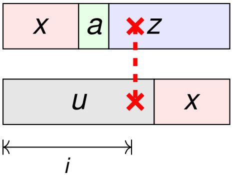


- they are satisfied by "existence of an interesting position"
- e.g.,

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

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

Why "Position Constraints"?



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

$$
\begin{aligned}
xaz \neq ux \quad &\Leftrightarrow \quad \exists i \in \mathbb{N}\colon (xaz)[i] \neq (ux)[i] \\
\neg suffixof(axb, yzw) \quad &\Leftrightarrow \quad \exists i \in \mathbb{N}\colon i < |axb| \wedge (axb)[-i] \neq (yzw)[-i] \\
\neg contains(xya, zyw) \quad &\Leftrightarrow \quad \forall k\colon 0 \leq k \leq |zyw| - |xya| \Rightarrow \exists i\colon (xya)[i] \neq (zyw)[i+k]
\end{aligned}
$$

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 $PF(\mathcal{A}_{tag}) \land \varphi$ by an off-the-shelf LIA solver where φ is formula encoding semantics of a position constraint

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 $\mathbb{T}$:

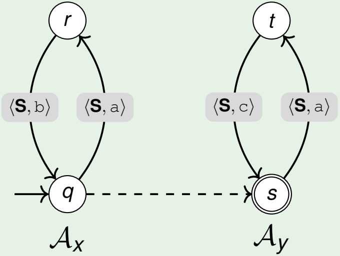
- extension of finite automaton
- $\mathcal{A}_{tag} = (Q, \Delta, I, F)$
  - Q: (finite) set of states
  - $I \subseteq Q$: initial states
  - $F \subseteq Q$: final states
  - $\Delta \subseteq Q \times 2^{\mathbb{T}} \times Q$: transitions

Example

$x \in (ab)^*$

$y \in (ac)^*$

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



Parikh formula

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 $\{\#\langle \mathbf{S}, a\rangle = 4, \#\langle \mathbf{S}, b\rangle = 2, \#\langle \mathbf{S}, c\rangle = 2\} \models PF(\mathcal{A}_{tag})$

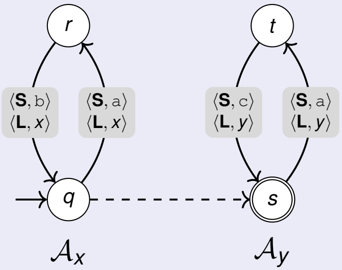
Length Constraints: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad 2|x| = 3|y| + 2$

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

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

$\mathcal{A}_{tag}$

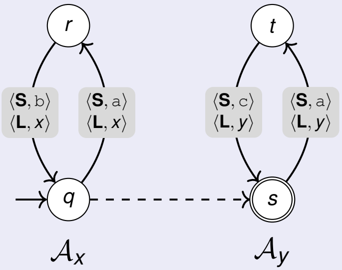


Length Constraints: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad 2|x| = 3|y| + 2$

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

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

$\mathcal{A}_{tag}$



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

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma : \mathbb{X} \to \Sigma^*$.

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma\colon \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma\colon \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

1. we are reading $\sigma(x)$, but we have not seen the interesting position

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma\colon \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

1. we are reading $\sigma(x)$, but we have not seen the interesting position
2. we see the letter at the interesting position in $\sigma(x)$

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma : \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

1. we are reading $\sigma(x)$, but we have not seen the interesting position
2. we see the letter at the interesting position in $\sigma(x)$
3. we finish reading $\sigma(x)$

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma : \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

1. we are reading $\sigma(x)$, but we have not seen the interesting position
2. we see the letter at the interesting position in $\sigma(x)$
3. we finish reading $\sigma(x)$
4. we start reading $\sigma(y)$

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma \colon \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

1. we are reading $\sigma(x)$, but we have not seen the interesting position
2. we see the letter at the interesting position in $\sigma(x)$
3. we finish reading $\sigma(x)$
4. we start reading $\sigma(y)$
5. we see the letter at the interesting position in $\sigma(y)$

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

Intuition

The run of $\mathcal{A}_{tag}$ maps to (possibly multiple) variable assignments $\sigma : \mathbb{X} \to \Sigma^*$.

Our construction captures the following sequence of event happening during a run:

1. we are reading $\sigma(x)$, but we have not seen the interesting position
2. we see the letter at the interesting position in $\sigma(x)$
3. we finish reading $\sigma(x)$
4. we start reading $\sigma(y)$
5. we see the letter at the interesting position in $\sigma(y)$
6. we finish reading $\sigma(x)$

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:

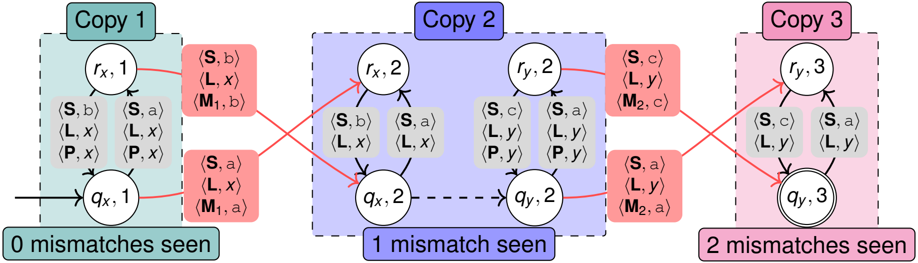
$$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\} \cup \{\langle \mathbf{P}, x\rangle, \langle \mathbf{P}, y\rangle\} \cup \{\langle \mathbf{M}_1, a\rangle, \langle \mathbf{M}_1,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

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

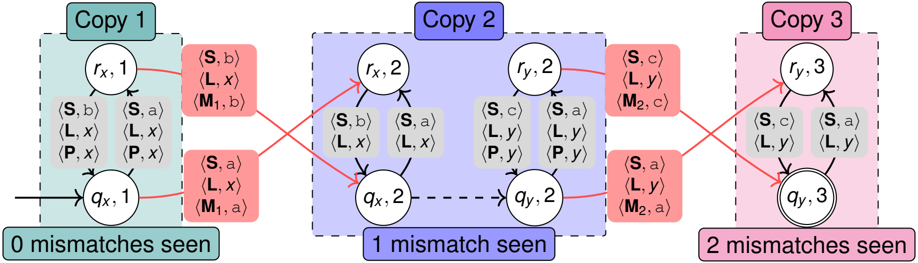


Single Simple Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad x \neq y$

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

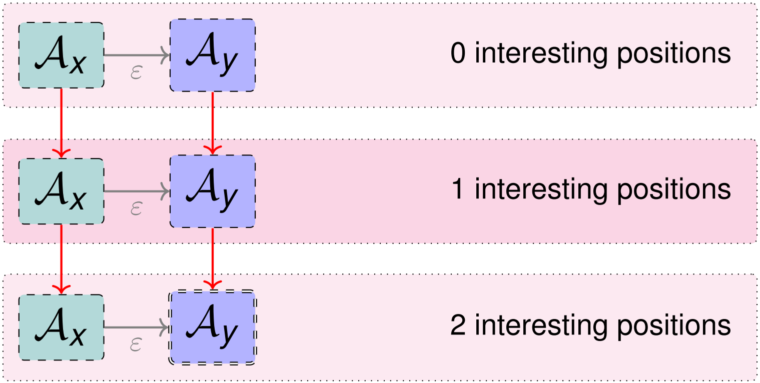
$$\mathbb{T} = \{\langle \mathbf{S}, a\rangle, \langle \mathbf{S}, b\rangle, \langle \mathbf{S}, c\rangle\} \cup \{\langle \mathbf{L}, x\rangle, \langle \mathbf{L}, y\rangle\} \cup \{\langle \mathbf{P}, x\rangle, \langle \mathbf{P}, y\rangle\} \cup \{\langle \mathbf{M}_1, a\rangle, \langle \mathbf{M}_1,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

- 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



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

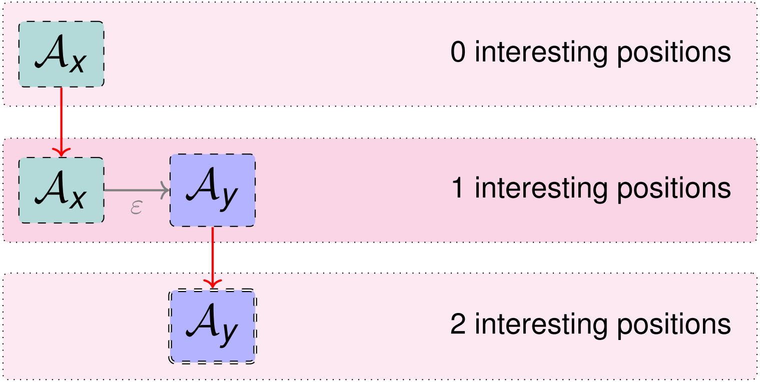
Zooming out



$\mathcal{A}_x$
ε
$\mathcal{A}_y$
0 interesting positions
$\mathcal{A}_x$
ε
$\mathcal{A}_y$
1 interesting positions
$\mathcal{A}_x$
ε
$\mathcal{A}_y$
2 interesting positions

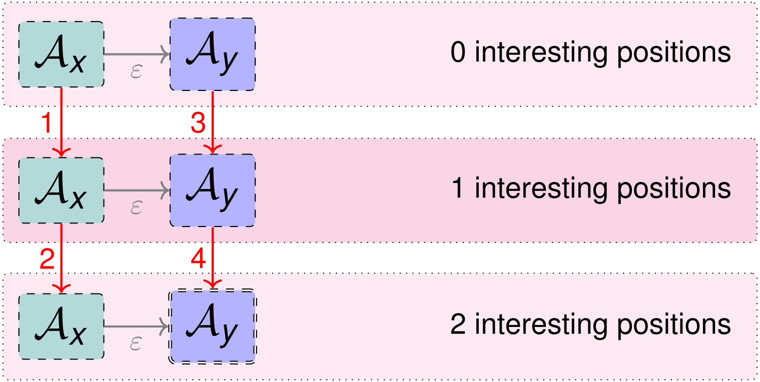
Zooming out

Considering the structure of the disequation $x \neq y$



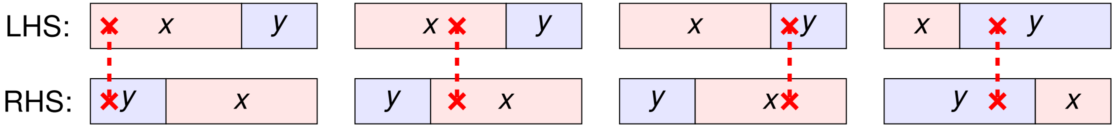
Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$

Which of the edges 1, 2, 3, 4 do we need?

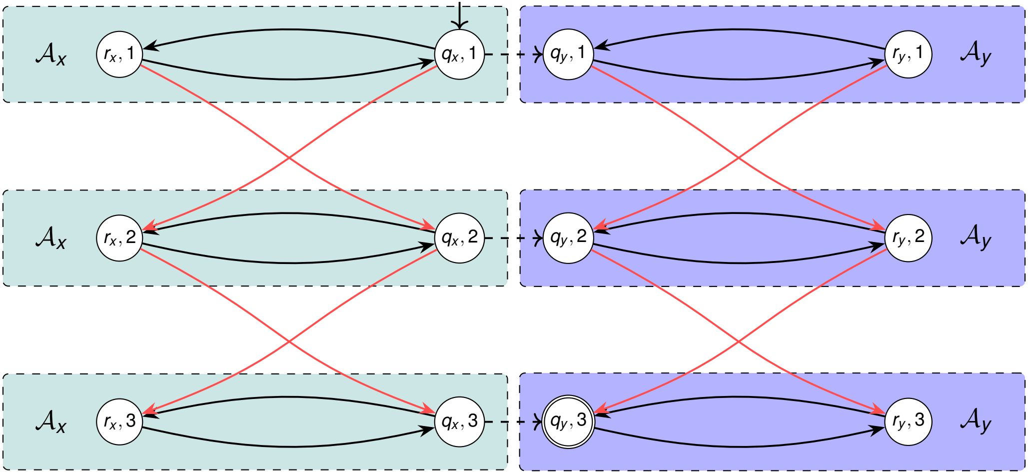


Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$

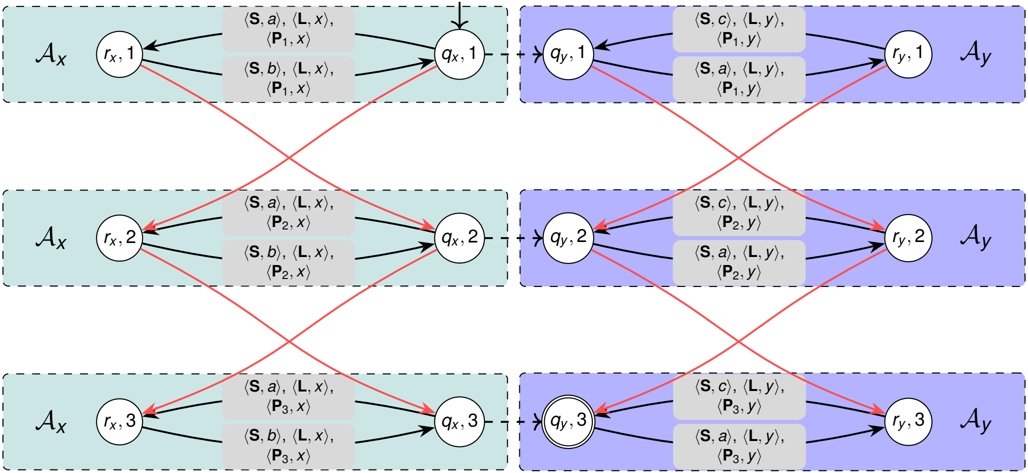
- now we need to consider several options:



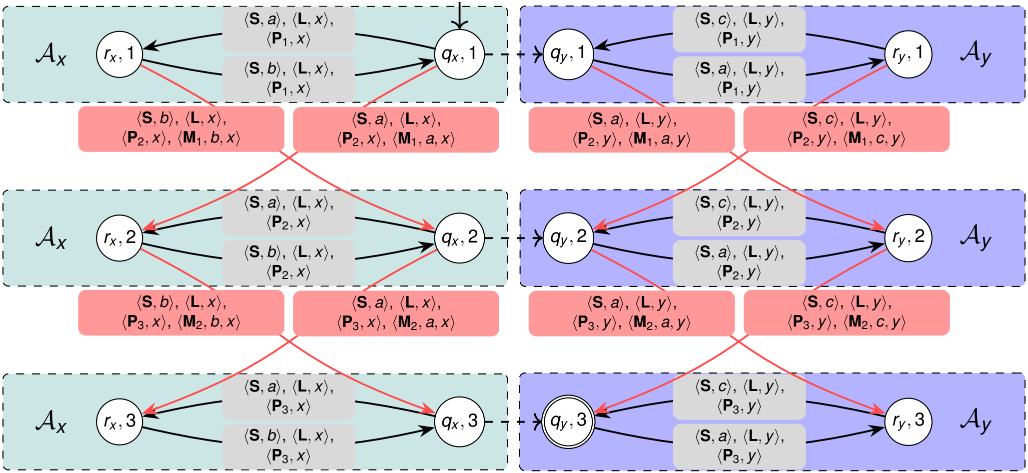
Single (not simple) Disequality: $x \in (ab)^* \land y \in (ac)^* \quad \land \quad xy \neq yx$



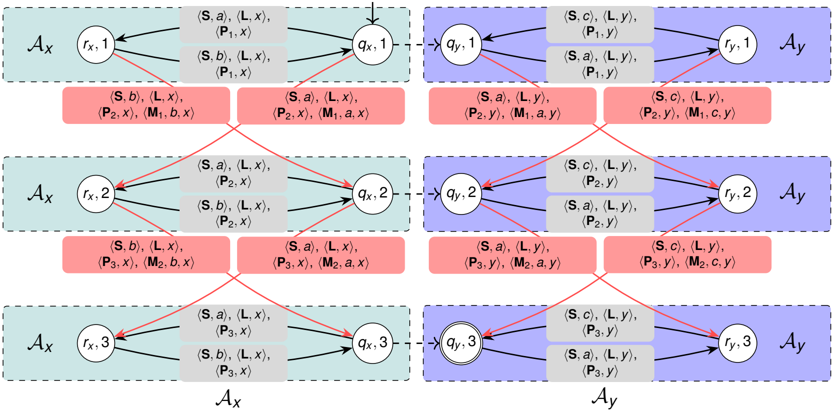
Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$



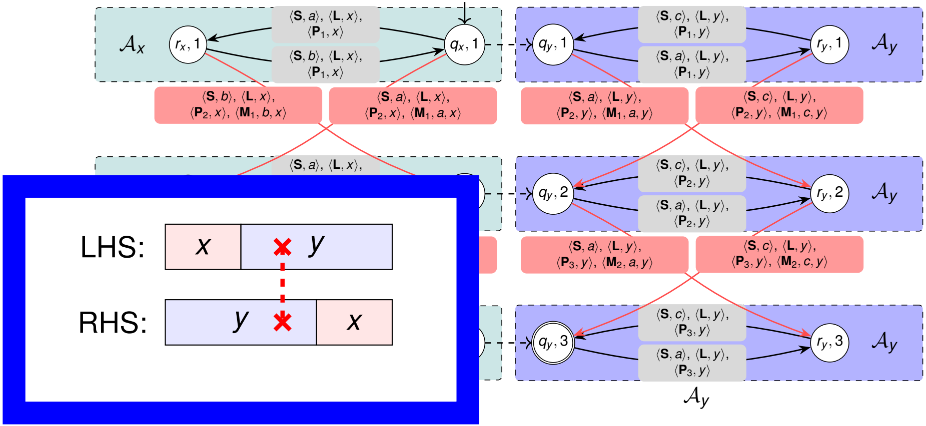
Single (not simple) Disequality: $x \in (ab)^* \land y \in (ac)^* \quad \land \quad xy \neq yx$



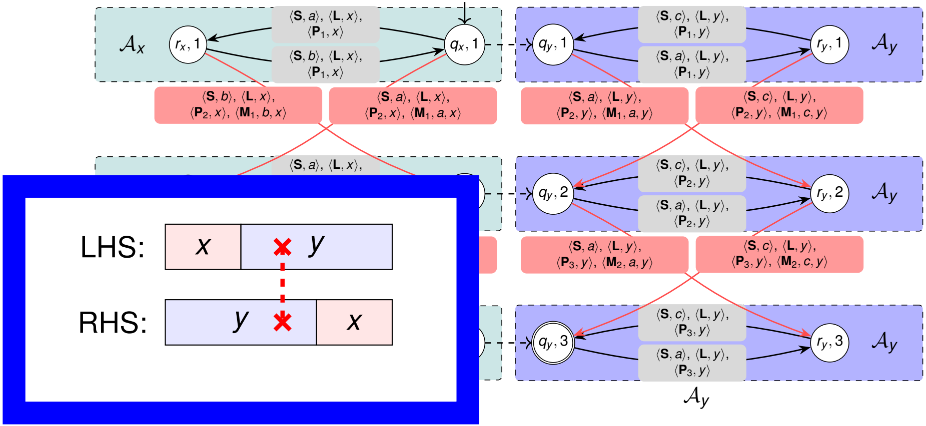
Single (not simple) Disequality: $x \in (ab)^* \wedge y \in (ac)^* \quad \wedge \quad xy \neq yx$



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



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



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

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

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

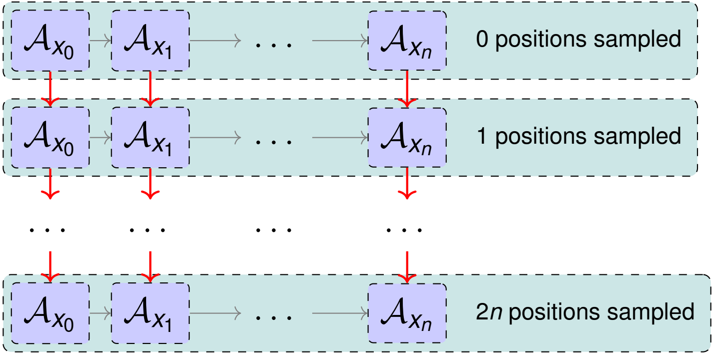
- we need to consider 2 positions (P_1, P_2) for $x \neq y$ and 2 positions (P_3, P_4) for $x \neq z$
- what is the order in which we will see P_1, P_2, P_3, P_4?
- naive solution: $\frac{(2n)!}{2^n} \in 2^{\Theta(n \log n)}$ (more copies of $\mathcal{A}_x$, $\mathcal{A}_y$)
  - $n \ldots$ number of disequalities

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

- we need to consider 2 positions (P_1, P_2) for $x \neq y$ and 2 positions (P_3, P_4) for $x \neq z$
- what is the order in which we will see P_1, P_2, P_3, P_4?
- naive solution: $\frac{(2n)!}{2^n} \in 2^{\Theta(n \log n)}$ (more copies of $\mathcal{A}_x$, $\mathcal{A}_y$)
  - $n \ldots$ number of disequalities
- better encoding: $\mathcal{O}(n)$

Better encoding for n disequations

We need to see/sample $2n$ interesting positions $\rightsquigarrow$ $2n + 1$ copies of the input automata.



Better encoding for n disequations

A single variable might occur in multiple disequalities

Consider

$$x \neq z \wedge x \neq y$$

Better encoding for n disequations

A single variable might occur in multiple disequalities

Consider

$$x \neq z \wedge x \neq y$$

The same interesting position in x might be a satisfiability witness in both disequations.

Better encoding for n disequations

A single variable might occur in multiple disequalities

Consider

$$x \neq z \wedge x \neq y$$

The same interesting position in x might be a satisfiability witness in both disequations.

We allow ε transitions between levels, labeled with a ‘copy’ tag that represents that the interesting position is the same as the previous one.

Better encoding for *n* disequations

Two kinds of tags labeling sampling transitions

1. $\langle \mathbf{M}_i, x, D, s, \mathtt{a} \rangle$ - mismatch sampling transition

Better encoding for n disequations

Two kinds of tags labeling sampling transitions

1. $\langle \mathbf{M}_i, x, D, s, \mathtt{a} \rangle$ - mismatch sampling transition
   - We are seeing i-th interesting position,
   - in variable x,
   - for the disequation D and its side $s \in \{\mathcal{L}\text{eft}, \mathcal{R}\text{ight}\}$, and
   - the symbol is $\mathtt{a}$.

Better encoding for n disequations

Two kinds of tags labeling sampling transitions

1. $\langle \mathbf{M}_i, x, D, s, \mathtt{a} \rangle$ - mismatch sampling transition
   - We are seeing i-th interesting position,
   - in variable x,
   - for the disequation D and its side $s \in \{\mathcal{L}\text{eft}, \mathcal{R}\text{ight}\}$, and
   - the symbol is $\mathtt{a}$.
2. $\langle \mathbf{C}_i, x, D, s \rangle$ - copy the previous sampling transition

Better encoding for n disequations

Two kinds of tags labeling sampling transitions

1. $\langle \mathbf{M}_i, x, D, s, \mathtt{a} \rangle$ - mismatch sampling transition
   - We are seeing i-th interesting position,
   - in variable x,
   - for the disequation D and its side $s \in \{\mathcal{L}\text{eft}, \mathcal{R}\text{ight}\}$, and
   - the symbol is $\mathtt{a}$.
2. $\langle \mathbf{C}_i, x, D, s \rangle$ - copy the previous sampling transition

We introduce two auxiliary variables allowing 'inductive' result formula:

1. $m_{D,s}$ - the value of the sampled symbol for the disequation D and its side $s \in \{\mathcal{L}\text{eft}, \mathcal{R}\text{ight}\}$
2. c_i - the i-th interesting symbol

Better encoding for n disequations

Resulting formula

$$\varphi_{Consistent} \overset{\text{def.}}{\Leftrightarrow} \bigwedge_{\substack{D \in \{D_1,\ldots,D_n\} \\ s \in \{\mathcal{L},\mathcal{R}\}, \mathtt{a} \in \Gamma, \\ 1 \le i \le 2n}} \left(\left(\sum_{x \in \mathbb{X}} \#\langle \mathbf{M}_i, x, D, s, a \rangle = 1 \right) \rightarrow c_i = m_{D,s} = \mathtt{a} \right) \land \left(\left(\sum_{x \in \mathbb{X}} \#\langle \mathbf{C}_i, x, D, s \rangle = 1 \right) \rightarrow c_i = m_{D,s} = c_{i-1} \right)$$

Better encoding for n disequations

Resulting formula

$$\varphi_{Consistent} \overset{\text{def.}}{\Leftrightarrow} \bigwedge_{\substack{D\in\{D_1,\ldots,D_n\}\\ s\in\{\mathcal{L},\mathcal{R}\},\mathtt{a}\in\Gamma,\\ 1\leq i\leq 2n}} \left(\left(\sum_{x\in\mathbb{X}} \#\langle \mathbf{M}_i, x, D, s, a\rangle = 1\right) \rightarrow c_i = m_{D,s} = \mathtt{a}\right) \land$$

$$\left(\left(\sum_{x\in\mathbb{X}} \#\langle \mathbf{C}_i, x, D, s\rangle = 1\right) \rightarrow c_i = m_{D,s} = c_{i-1}\right)$$

$$\psi \overset{\text{def.}}{\Leftrightarrow} PF(\mathcal{A}_{Tag}) \land \varphi_{Fair} \land \varphi_{Consistent} \land \varphi_{Copies} \land \bigwedge_{D\in\{D_1,\ldots,D_n\}} (\varphi^D_{len} \lor (\varphi^D_{mis} \land \varphi^D_{sym}))$$

Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \land x_1 \in \ldots \land x_n \in \ldots$

Recall $\neg contains(\mathcal{N}, \mathcal{H})$ is satisfied by σ if $\sigma(\mathcal{N})$ is not a substring of $\sigma(\mathcal{H})$

- $\mathcal{N}$eedle is not in the $\mathcal{H}$aystack

Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \wedge x_1 \in \ldots \wedge x_n \in \ldots$

Recall $\neg contains(\mathcal{N}, \mathcal{H})$ is satisfied by σ if $\sigma(\mathcal{N})$ is not a substring of $\sigma(\mathcal{H})$

- $\mathcal{N}$eedle is not in the $\mathcal{H}$aystack

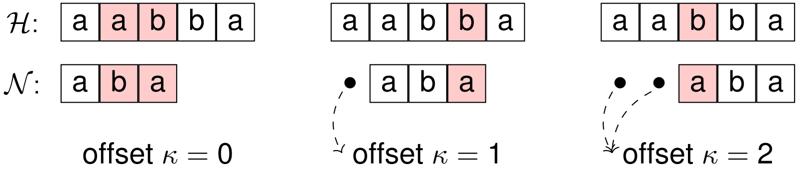
Implicit universal quantification $\neg contains(\mathcal{N}, \mathcal{H}) \Leftrightarrow \forall p, s(p\mathcal{N}s \neq \mathcal{H})$

Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \wedge x_1 \in \ldots \wedge x_n \in \ldots$

Recall $\neg contains(\mathcal{N}, \mathcal{H})$ is satisfied by σ if $\sigma(\mathcal{N})$ is not a substring of $\sigma(\mathcal{H})$

- $\mathcal{N}$eedle is not in the $\mathcal{H}$aystack

Implicit universal quantification $\neg contains(\mathcal{N}, \mathcal{H}) \Leftrightarrow \forall p, s(p\mathcal{N}s \neq \mathcal{H})$

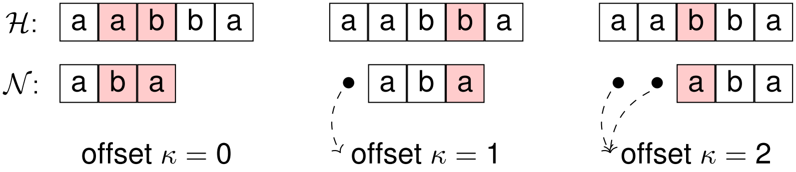


Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \wedge x_1 \in \ldots \wedge x_n \in \ldots$

Recall $\neg contains(\mathcal{N}, \mathcal{H})$ is satisfied by σ if $\sigma(\mathcal{N})$ is not a substring of $\sigma(\mathcal{H})$

- $\mathcal{N}$eedle is not in the $\mathcal{H}$aystack

Implicit universal quantification $\neg contains(\mathcal{N}, \mathcal{H}) \Leftrightarrow \forall p, s(p\mathcal{N}s \neq \mathcal{H})$



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

Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \wedge x_1 \in \ldots \wedge x_n \in \ldots$

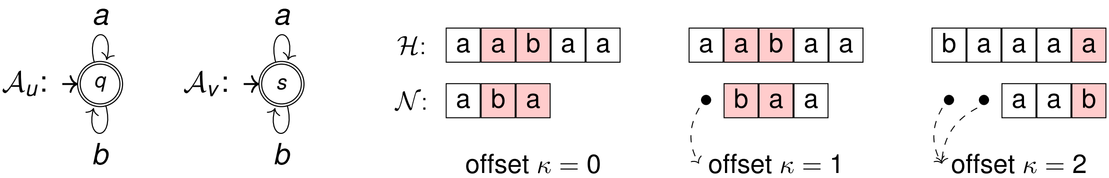
$$\forall \kappa \geq 0 \colon \exists \#\delta_1, \ldots \#\delta_n \colon PF(\mathcal{A}_{tag}) \wedge \ldots$$

- 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!

Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \land x_1 \in \ldots \land x_n \in \ldots$

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

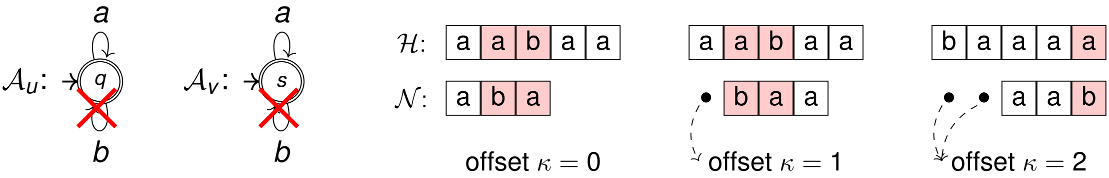
- universal quantification of the offset κ doesn't work!
- one model of $PF(\mathcal{A}_{tag})$ may correspond to several different words
- $\rightsquigarrow$ allows different models for different κ!
- example: suppose $\mathcal{H} = v, \mathcal{N} = u, m = \{\#a \mapsto 4, \#b \mapsto 1\}$



Not Contains: $\neg contains(\mathcal{N}, \mathcal{H}) \land x_1 \in \ldots \land x_n \in \ldots$

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

- 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 $\mathcal{H} = v, \mathcal{N} = u, m = \{\#a \mapsto 4, \#b \mapsto 1\}$



- restriction to flat regular constraints
  - ▶ $\forall\exists$ LIA formula

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 $\ldots \mathcal{O}(nm \cdot |\Sigma|^3 \cdot |\mathcal{R}|^6)$
  - also for $\neg$*prefixof*, $\neg$*suffixof*

Decidability and Complexity

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

- REG with single disequality $x_1 \dots x_m \neq y_1 \dots y_n$:
  - 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

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 ... $\mathcal{O}(nm \cdot |\Sigma|^3 \cdot |\mathcal{R}|^6)$
  - 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*

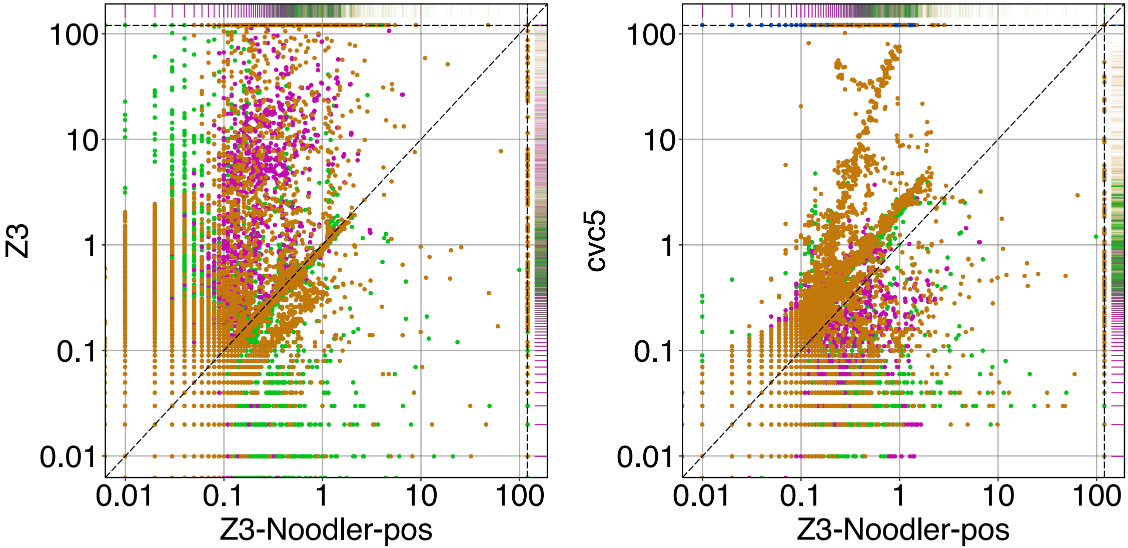
[1] Abdulla et al. “Solving not-substring constraint with flat abstraction”. In: *APLAS’21*.

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

Position constraints
Conclusion

Takeaway:

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

Part 2: Negated String Containment is Decidable

(Published at MFCS'25)


This is where the fun begins.

Problem statement

and performed normalization

NEGATED STRING CONTAINMENT	
Input:	A formula $\varphi \triangleq \neg \mathit{contains}(\mathcal{N}, \mathcal{H}) \wedge \bigwedge_{x \in \mathbb{X}} x \in L_x$ with $\mathcal{N}, \mathcal{H} \in (\Sigma \cup \mathbb{X})^*$
Question:	Find a morphism $\sigma\colon \mathbb{X} \to \Sigma^*$ such that $\sigma(x) \in L_x$ for every variable x and $\sigma(\mathcal{N})$ is not a factor $\sigma(\mathcal{H})$

- intuition: $\mathcal{N}$eedle is not withing the $\mathcal{H}$aystack

Problem statement

and performed normalization

NEGATED STRING CONTAINMENT

Input: A formula $\varphi \triangleq \neg contains(\mathcal{N}, \mathcal{H}) \wedge \bigwedge_{x \in \mathbb{X}} x \in L_x$ with $\mathcal{N}, \mathcal{H} \in (\Sigma \cup \mathbb{X})^*$

Question: Find a morphism $\sigma \colon \mathbb{X} \to \Sigma^*$ such that $\sigma(x) \in L_x$ for every variable x and $\sigma(\mathcal{N})$ is not a factor $\sigma(\mathcal{H})$

- intuition: $\mathcal{N}$eedle is not withing the $\mathcal{H}$aystack

Step 0. Normalization

Results in a disjunction of $\bigvee_{i \in I} \neg contains(\mathcal{N}_i, \mathcal{H}_i) \wedge \varphi_{L,i}$ where each $\neg contains(\mathcal{N}_i, \mathcal{H}_i) \wedge \varphi_{L,i}$ satisfies:

- every flat variable x has a language w_x^* for some $w_x \in \Sigma^*$,
- every non-flat variable y has a language S_y^* for some set of words $S_y \subseteq \Sigma^*$.

Problem statement

and performed normalization

Negated String Containment	
Input:	A formula $\varphi \triangleq \neg \mathit{contains}(\mathcal{N}, \mathcal{H}) \wedge \bigwedge_{x \in \mathbb{X}} x \in L_x$ with $\mathcal{N}, \mathcal{H} \in (\Sigma \cup \mathbb{X})^*$
Question:	Find a morphism $\sigma \colon \mathbb{X} \to \Sigma^*$ such that $\sigma(x) \in L_x$ for every variable x and $\sigma(\mathcal{N})$ is not a factor $\sigma(\mathcal{H})$

- intuition: $\mathcal{N}$eedle is not withing the $\mathcal{H}$aystack

Step 0. Normalization

Results in a disjunction of $\bigvee_{i \in I} \neg \mathit{contains}(\mathcal{N}_i, \mathcal{H}_i) \wedge \varphi_{L,i}$ where each $\neg \mathit{contains}(\mathcal{N}_i, \mathcal{H}_i) \wedge \varphi_{L,i}$ satisfies:

- every flat variable x has a language w_x^* for some $w_x \in \Sigma^*$,
- every non-flat variable y has a language S_y^* for some set of words $S_y \subseteq \Sigma^*$.

We need to decide satisfiability of individual disjuncts.

When is $\neg$*contains*$(\mathcal{N}, \mathcal{H})$ easy?

- we can find $\sigma : \mathbb{X} \to \Sigma^*$ such that $|\sigma(\mathcal{N})| > |\sigma(\mathcal{H})|$,
- all variables are flat, or
- $\mathcal{N}$ is a literal.

When is $\neg$*contains*$(\mathcal{N}, \mathcal{H})$ easy?

- we can find $\sigma : \mathbb{X} \to \Sigma^*$ such that $|\sigma(\mathcal{N})| > |\sigma(\mathcal{H})|$,
- all variables are flat, or
- $\mathcal{N}$ is a literal.

What situations we cannot solve?

When is $\neg contains(\mathcal{N}, \mathcal{H})$ easy?

- we can find $\sigma : \mathbb{X} \to \Sigma^*$ such that $|\sigma(\mathcal{N})| > |\sigma(\mathcal{H})|$,
- all variables are flat, or
- $\mathcal{N}$ is a literal.

What situations we cannot solve?

Every non-flat variable $x \in \mathbb{X}$ satisfies:

1. $\#_x(\mathcal{N}) = 0 \wedge \#_x(\mathcal{H}) > 0$, or
   - x occurs only in $\mathcal{H}$

When is $\neg contains(\mathcal{N}, \mathcal{H})$ easy?

- we can find $\sigma : \mathbb{X} \to \Sigma^*$ such that $|\sigma(\mathcal{N})| > |\sigma(\mathcal{H})|$,
- all variables are flat, or
- $\mathcal{N}$ is a literal.

What situations we cannot solve?

Every non-flat variable $x \in \mathbb{X}$ satisfies:

1. $\#_x(\mathcal{N}) = 0 \wedge \#_x(\mathcal{H}) > 0$, or
   - x occurs only in $\mathcal{H}$
2. $\#_x(\mathcal{H}) > \#_x(\mathcal{N}) > 0$.
   - x occurs on both sides

Step 1. How to handle non-flat variables occurring both in $\mathcal{H}$ and $\mathcal{N}$?

Dealing with situations when x is on both sides

Our input formula is

$$\neg \mathit{contains}(\mathcal{N}, \mathcal{H}) \wedge \varphi_L$$

Assume that we have a partial assignment $\sigma' \colon \mathbb{X} \setminus \{x\} \to \Sigma^*$, such that

$$\sigma' \models \neg \mathit{contains}(\mathcal{N}', \mathcal{H}')$$

where $\neg \mathit{contains}(\mathcal{N}', \mathcal{H}') \triangleq \neg \mathit{contains}(\mathcal{N}[x/\diamond_x], \mathcal{H}[x/\diamond_x])$.

Dealing with situations when x is on both sides

Our input formula is

$$\neg contains(\mathcal{N}, \mathcal{H}) \wedge \varphi_L$$

Assume that we have a partial assignment $\sigma' \colon \mathbb{X} \setminus \{x\} \to \Sigma^*$, such that

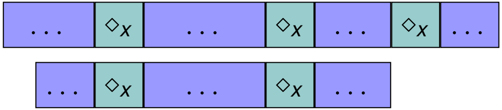
$$\sigma' \models \neg contains(\mathcal{N}', \mathcal{H}')$$

where $\neg contains(\mathcal{N}', \mathcal{H}') \triangleq \neg contains(\mathcal{N}[x/\diamond_x], \mathcal{H}[x/\diamond_x])$.

Intuitively, σ' is interesting only when $\diamond_x$ in $\sigma'(\mathcal{H}[x/\diamond_x])$ is above some $\diamond_x$ in $\sigma'(\mathcal{H}[x/\diamond_x])$.

$\sigma'(\mathcal{H}[x/\diamond_x])$

$\sigma'(\mathcal{N}[x/\diamond_x])$



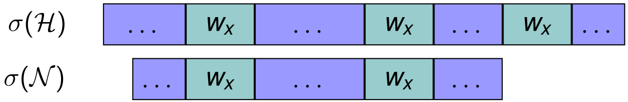
Dealing with situations when x is on both sides

Let $w_x \in L_x$, and let $\sigma \triangleq \sigma' \triangleleft \{x \mapsto w_x\}$.

Dealing with situations when x is on both sides

Let $w_x \in L_x$, and let $\sigma \triangleq \sigma' \triangleleft \{x \mapsto w_x\}$.

Since $\sigma' \models \neg contains(\mathcal{N}', \mathcal{H}')$ we know that

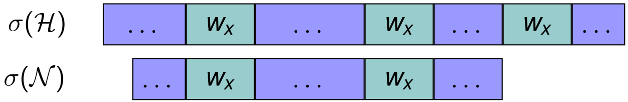


contains a conflict.

Dealing with situations when x is on both sides

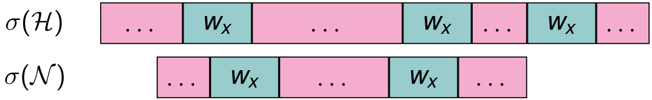
Let $w_x \in L_x$, and let $\sigma \triangleq \sigma' \triangleleft \{x \mapsto w_x\}$.

Since $\sigma' \models \neg \mathit{contains}(\mathcal{N}', \mathcal{H}')$ we know that



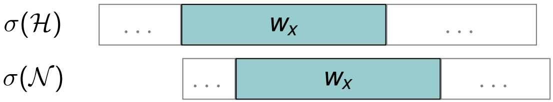
contains a conflict.

Therefore, if $\sigma \not\models \neg \mathit{contains}(\mathcal{N}, \mathcal{H})$, we cannot have every w_x in $\sigma(\mathcal{N})$ under some w_x in $\sigma(\mathcal{H})$.



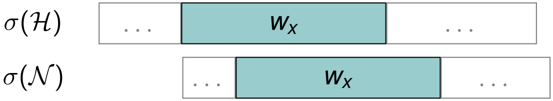
Dealing with situations when x is on both sides

By picking long enough w_x we force that if $\sigma \not\models \neg contains(\mathcal{N}, \mathcal{H})$, then we must have w_x from $\sigma(\mathcal{N})$ partially overlapping with some w_x from $\sigma(\mathcal{H})$.



Dealing with situations when x is on both sides

By picking long enough w_x we force that if $\sigma \not\models \neg contains(\mathcal{N}, \mathcal{H})$, then we must have w_x from $\sigma(\mathcal{N})$ partially overlapping with some w_x from $\sigma(\mathcal{H})$.



All that is needed is to come up with a special w_x that cannot have conflict-free overlaps (of sufficient size) with itself, which would allow us to *always* construct a model σ from σ'.

Dealing with situations when *x* is on both sides

But to a get a decision procedure, we don't have the initial partial assignment σ'!

Dealing with situations when x is on both sides

But to a get a decision procedure, we don't have the initial partial assignment σ'!

Just replace x with a fresh symbol $\diamond_x$, producing $\neg contains(\mathcal{N}', \mathcal{H}')$ and removing the two-sided variable x.

Dealing with situations when x is on both sides

But to a get a decision procedure, we don't have the initial partial assignment σ'!

Just replace x with a fresh symbol $\diamond_x$, producing $\neg contains(\mathcal{N}', \mathcal{H}')$ and removing the two-sided variable x.

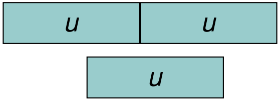
If we can solve $\neg contains(\mathcal{N}', \mathcal{H}')$, then we can replace $\diamond_x$ with a suitable w_x as described above.

Enter combinatorics on words (CoW)

How to choose w_x with the desired properties

A word u is called *primitive* if $u \notin w^*$ for any word $w \neq u$.

Primitive words have cool properties, e.g., if $uu = pus$, then either $p = \varepsilon$ or $s = \varepsilon$. Graphically, the following is not possible.



CoW to the rescue

Thanks to our normalization, we have $\{u, v\}^* \subseteq L_x$ with $u, v \notin w^*$ for any word w.

[2] Lyndon & Schützenberger: The equation $a^M = b^N c^P$ in a free group.

CoW to the rescue

Thanks to our normalization, we have $\{u, v\}^* \subseteq L_x$ with $u, v \notin w^*$ for any word w.

Let us define α and β as

$$\alpha \triangleq u^2 u^k v^2 \quad \in L_x$$
$$\beta \triangleq u^2 v^l v^2 \quad \in L_x$$

for $k = \mathrm{lcm}(|u|, |v|)/|u|$ and $l = \mathrm{lcm}(|u|, |v|)/|v|$.

- Thanks to the results of Lyndon and Schützenberger[2], we have both α and β primitive.

[2] Lyndon & Schützenberger: The equation $a^M = b^N c^P$ in a free group.

CoW to the rescue

Thanks to our normalization, we have $\{u, v\}^* \subseteq L_x$ with $u, v \notin w^*$ for any word w.

Let us define α and β as

$$\alpha \triangleq u^2 u^k v^2 \quad \in L_x$$
$$\beta \triangleq u^2 v^l v^2 \quad \in L_x$$

for $k = \mathrm{lcm}(|u|, |v|)/|u|$ and $l = \mathrm{lcm}(|u|, |v|)/|v|$.

- Thanks to the results of Lyndon and Schützenberger[2], we have both α and β primitive.

Finally, the word

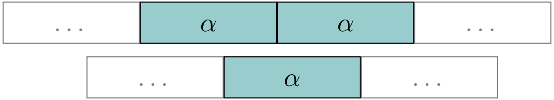
$$w_x \triangleq \alpha^M \beta^M \alpha^M \beta^M \alpha^{2M} \beta^{2M}$$

prevents large overlaps with itself, where $M \in \mathbb{N}$ is chosen so that w_x is sufficiently large.

[2]Lyndon & Schützenberger: The equation $a^M = b^N c^P$ in a free group.

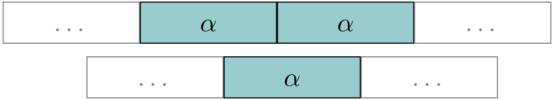
Ehm, what?

To show that w_x truly has the desired properties we first observe that whenever we consider a long enough overlap, we have α^2 above α (or similarly for β).



Ehm, what?

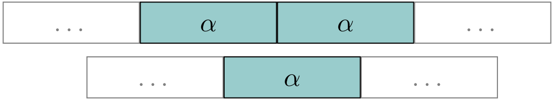
To show that w_x truly has the desired properties we first observe that whenever we consider a long enough overlap, we have α^2 above α (or similarly for β).



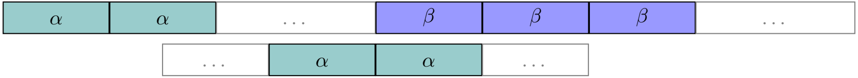
Therefore, we need to consider overlaps of w_x with w_x only with certain granularity.

Ehm, what?

To show that w_x truly has the desired properties we first observe that whenever we consider a long enough overlap, we have α^2 above α (or similarly for β).

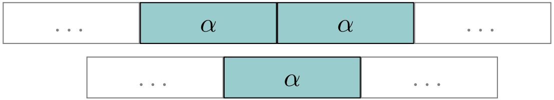


Therefore, we need to consider overlaps of w_x with w_x only with certain granularity.

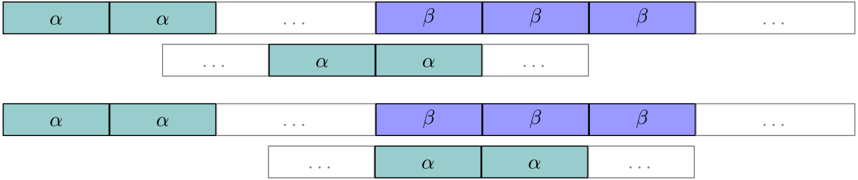


Ehm, what?

To show that w_x truly has the desired properties we first observe that whenever we consider a long enough overlap, we have α^2 above α (or similarly for β).

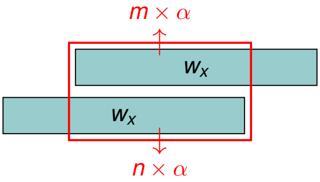


Therefore, we need to consider overlaps of w_x with w_x only with certain granularity.



Ehm, what?

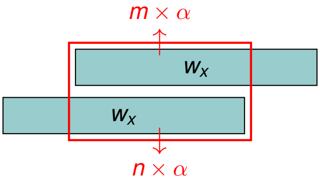
For any of such remaining ‘granular’ overlaps we directly show that whenever we consider an overlap of w_x with itself, there is a different number of α’s in the overlapping portions of w_x from $\sigma(\mathcal{N})$ and $\sigma(\mathcal{H})$[3].



[3] except in one case

Ehm, what?

For any of such remaining 'granular' overlaps we directly show that whenever we consider an overlap of w_x with itself, there is a different number of α's in the overlapping portions of w_x from $\sigma(\mathcal{N})$ and $\sigma(\mathcal{H})$[3].



So, we are left with the problem of having only flat variables in $\mathcal{N}$.

[3] except in one case

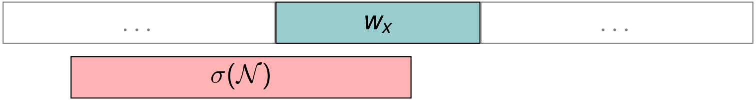
Non-flat variables present only in $\mathcal{H}$

Again, assume a partial assignment $\sigma' \colon \mathbb{X} \setminus \{x\} \to \Sigma^*$.

Non-flat variables present only in $\mathcal{H}$

Again, assume a partial assignment $\sigma' \colon \mathbb{X} \setminus \{x\} \to \Sigma^*$.

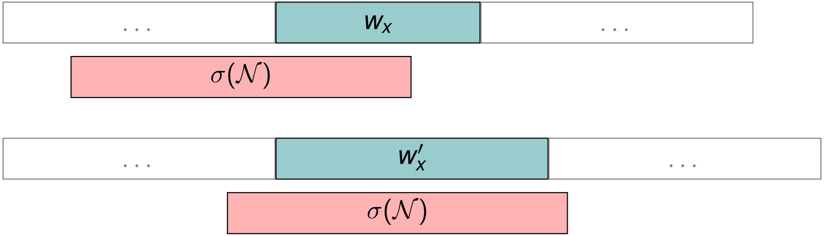
A naive approach would be to enumerate $w \in L_x$, and check whether $\sigma \triangleq \sigma' \triangleleft \{x \mapsto w\}$ is a model.



Non-flat variables present only in $\mathcal{H}$

Again, assume a partial assignment $\sigma' \colon \mathbb{X} \setminus \{x\} \to \Sigma^*$.

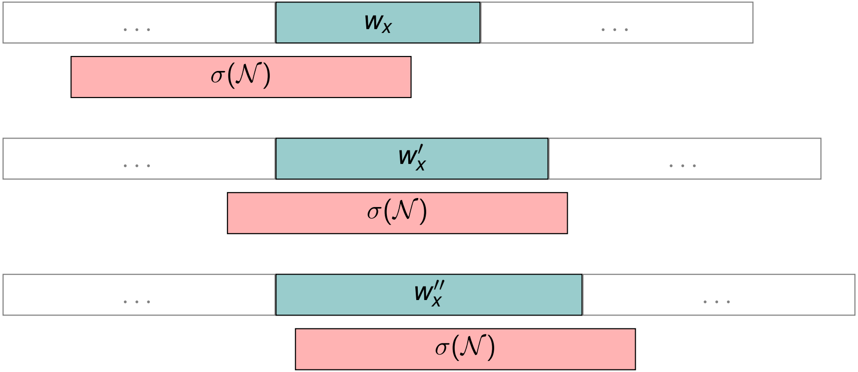
A naive approach would be to enumerate $w \in L_x$, and check whether $\sigma \triangleq \sigma' \triangleleft \{x \mapsto w\}$ is a model.



Non-flat variables present only in $\mathcal{H}$

Again, assume a partial assignment $\sigma' : \mathbb{X} \setminus \{x\} \to \Sigma^*$.

A naive approach would be to enumerate $w \in L_x$, and check whether $\sigma \triangleq \sigma' \triangleleft \{x \mapsto w\}$ is a model.



Non-flat variables present only in $\mathcal{H}$

It is problematic to know why σ fails to be a model.

Non-flat variables present only in $\mathcal{H}$

It is problematic to know why σ fails to be a model.

If would be much easier to solve a modified formulae

$$\varphi_{Pref} \triangleq \neg contains(\mathcal{N}[x/x_p\#], \mathcal{H}[x/x_p\#]),$$
$$\varphi_{Suf} \triangleq \neg contains(\mathcal{N}[x/\#x_s], \mathcal{H}[x/\#x_s]).$$

where $\#$ is a fresh separator symbol and x_p (x_s) is restricted to prefixes (suffixes) of x.

Non-flat variables present only in $\mathcal{H}$

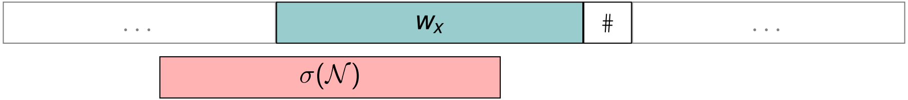
It is problematic to know why σ fails to be a model.

If would be much easier to solve a modified formulae

$$\varphi_{Pref} \triangleq \neg contains(\mathcal{N}[x/x_p\#], \mathcal{H}[x/x_p\#]),$$
$$\varphi_{Suf} \triangleq \neg contains(\mathcal{N}[x/\#x_s], \mathcal{H}[x/\#x_s]).$$

where $\#$ is a fresh separator symbol and x_p (x_s) is restricted to prefixes (suffixes) of x.

Intuitively, if $\sigma \not\models \varphi_{Pref}$ then we have the following situation:



Modularizing the proof

We introduce Γ_x—a tool that allows us to solve φ_{Pref} and φ_{Suf} separately[4] and then glue together the prefix and suffix to produce $\sigma \models \neg \mathit{contains}(\mathcal{N}, \mathcal{H})$.

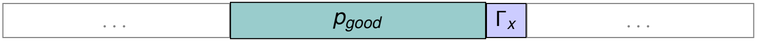
- Γ_x is an infix that acts as a fresh separator symbol $\#$

[4]With some technical assumptions on σ'

Modularizing the proof

We introduce Γ_x—a tool that allows us to solve φ_{Pref} and φ_{Suf} separately[4] and then glue together the prefix and suffix to produce $\sigma \models \neg contains(\mathcal{N}, \mathcal{H})$.

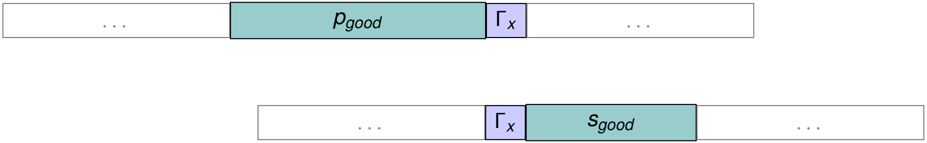
- Γ_x is an infix that acts as a fresh separator symbol $\#$



Modularizing the proof

We introduce Γ_x—a tool that allows us to solve φ_{Pref} and φ_{Suf} separately[4] and then glue together the prefix and suffix to produce $\sigma \models \neg contains(\mathcal{N}, \mathcal{H})$.

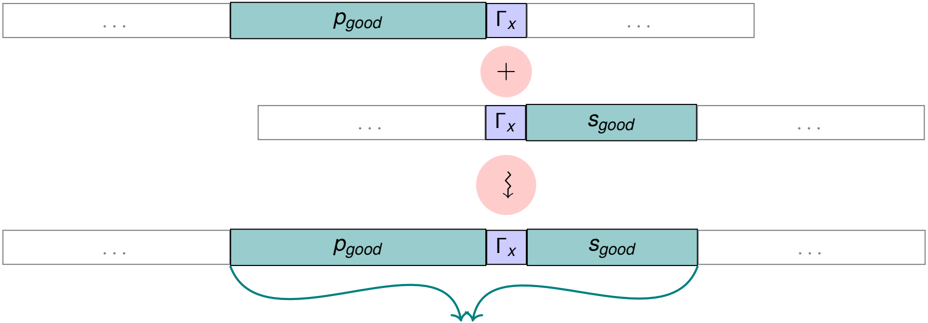
- Γ_x is an infix that acts as a fresh separator symbol $\#$



Modularizing the proof

We introduce Γ_x—a tool that allows us to solve φ_{Pref} and φ_{Suf} separately[4] and then glue together the prefix and suffix to produce $\sigma \models \neg contains(\mathcal{N}, \mathcal{H})$.

- Γ_x is an infix that acts as a fresh separator symbol $\#$

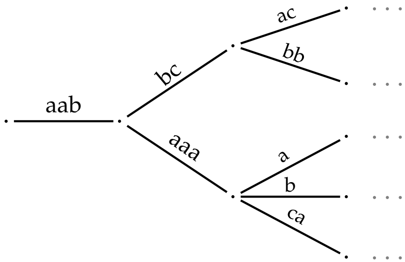


$$\sigma' \triangleleft \{x \mapsto p\#s\} \models \neg contains(\mathcal{N}, \mathcal{H})$$

[4]With some technical assumptions on σ'

Finding a suitable prefix

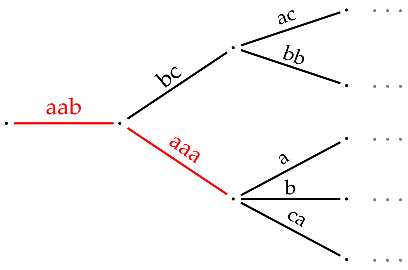
We explore prefixes of x systematically, using a *prefix tree*.



Finding a suitable prefix

Some vertices are dead ends

Consider the prefix aabaaa, and the following situation.

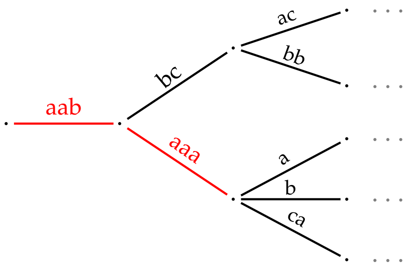


$\sigma(\mathcal{H})$	...	c	c	a	a	b	a	a	a	Γ_x	...
$\sigma(\mathcal{N})$		c	c	a	a	b	a				

Finding a suitable prefix

Some vertices are dead ends

Consider the prefix aabaaa, and the following situation.



$\sigma(\mathcal{H})$	...	c	c	a	a	b	a	a	a	Γ_x	...
$\sigma(\mathcal{N})$		c	c	a	a	b	a				

We mark some nodes as dead ends, and do not explore their successors.

Core theorem

Theorem (Simplified)

Given a $\neg$contains formula φ and a non-flat variable x occurring only in $\mathcal{H}$, we can compute a flat language $L'_x \subseteq L_x$ such that $\varphi \wedge x \in L'_x$ and φ are equisatisfiable.

Proof sketch.

1. construct $S_\Gamma = \{p\Gamma_x \mid p \in \mathrm{Pref}(L_x) \wedge |p| < \lambda\}$
2. show that if there is a model such that $|\sigma(x)| > \lambda$ but S_Γ contains no values that yield a model, then $\sigma(x)$ must have a special structure

Core theorem

Theorem (Simplified)

Given a $\neg$contains formula φ and a non-flat variable x occurring only in $\mathcal{H}$, we can compute a flat language $L'_x \subseteq L_x$ such that $\varphi \wedge x \in L'_x$ and φ are equisatisfiable.

Proof sketch.

1. construct $S_\Gamma = \{p\Gamma_x \mid p \in \mathrm{Pref}(L_x) \wedge |p| < \lambda\}$
2. show that if there is a model such that $|\sigma(x)| > \lambda$ but S_Γ contains no values that yield a model, then $\sigma(x)$ must have a special structure
   - $\sigma(x)$ starts with $s_\alpha \alpha^k p_\alpha$ where the language of the rightmost variable in $\mathcal{N}$ is $(\alpha^\ell)^*$,

Core theorem

Theorem (Simplified)

Given a $\neg$contains formula φ and a non-flat variable x occurring only in $\mathcal{H}$, we can compute a flat language $L'_x \subseteq L_x$ such that $\varphi \land x \in L'_x$ and φ are equisatisfiable.

Proof sketch.

1. construct $S_\Gamma = \{p\Gamma_x \mid p \in \mathrm{Pref}(L_x) \land |p| < \lambda\}$
2. show that if there is a model such that $|\sigma(x)| > \lambda$ but S_Γ contains no values that yield a model, then $\sigma(x)$ must have a special structure
   - $\sigma(x)$ starts with $s_\alpha \alpha^k p_\alpha$ where the language of the rightmost variable in $\mathcal{N}$ is $(\alpha^\ell)^*$,
   - any alternative successor that would diverge from the $s_\alpha \alpha^k p_\alpha$ prefix leads to a dead end

Core theorem

Theorem (Simplified)

Given a $\neg$contains formula φ and a non-flat variable x occurring only in $\mathcal{H}$, we can compute a flat language $L'_x \subseteq L_x$ such that $\varphi \wedge x \in L'_x$ and φ are equisatisfiable.

Proof sketch.

1. construct $S_\Gamma = \{p\Gamma_x \mid p \in \mathrm{Pref}(L_x) \wedge |p| < \lambda\}$
2. show that if there is a model such that $|\sigma(x)| > \lambda$ but S_Γ contains no values that yield a model, then $\sigma(x)$ must have a special structure
   - $\sigma(x)$ starts with $s_\alpha \alpha^k p_\alpha$ where the language of the rightmost variable in $\mathcal{N}$ is $(\alpha^\ell)^*$,
   - any alternative successor that would diverge from the $s_\alpha \alpha^k p_\alpha$ prefix leads to a dead end
3. consequently, we construct a flat language of the form $s_\alpha \alpha^n p_\alpha \Gamma_z$

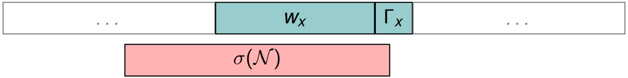
Core theorem

Theorem (Simplified)

Given a $\neg$contains formula φ and a non-flat variable x occurring only in $\mathcal{H}$, we can compute a flat language $L'_x \subseteq L_x$ such that $\varphi \land x \in L'_x$ and φ are equisatisfiable.

Proof sketch.

1. construct $S_\Gamma = \{p\Gamma_x \mid p \in \mathrm{Pref}(L_x) \land |p| < \lambda\}$
2. show that if there is a model such that $|\sigma(x)| > \lambda$ but S_Γ contains no values that yield a model, then $\sigma(x)$ must have a special structure
    - $\sigma(x)$ starts with $s_\alpha \alpha^k p_\alpha$ where the language of the rightmost variable in $\mathcal{N}$ is $(\alpha^\ell)^*$,
    - any alternative successor that would diverge from the $s_\alpha \alpha^k p_\alpha$ prefix leads to a dead end
3. consequently, we construct a flat language of the form $s_\alpha \alpha^n p_\alpha \Gamma_z$



Taking a step back

Entire decision procedure (sketch)

1. iteratively remove all non-flat variables that occur in both $\mathcal{N}$ and $\mathcal{H}$

Taking a step back

Entire decision procedure (sketch)

1. iteratively remove all non-flat variables that occur in both $\mathcal{N}$ and $\mathcal{H}$
2. replace languages of remaining non-flat variables with sound and complete flat underapproximations

Taking a step back

Entire decision procedure (sketch)

1. iteratively remove all non-flat variables that occur in both $\mathcal{N}$ and $\mathcal{H}$
2. replace languages of remaining non-flat variables with sound and complete flat underapproximations
3. solve resulting formula by reduction to LIA

Taking a step back

Entire decision procedure (sketch)

1. iteratively remove all non-flat variables that occur in both $\mathcal{N}$ and $\mathcal{H}$
2. replace languages of remaining non-flat variables with sound and complete flat underapproximations
3. solve resulting formula by reduction to LIA

Resulting complexity is $\textsc{ExpSpace}$.

Future work

- Extend the proof to conjunction of $\neg$*contains* predicates.
- Improve complexity bounds of the problem.

Future work

- Extend the proof to conjunction of $\neg$*contains* predicates.
- Improve complexity bounds of the problem.

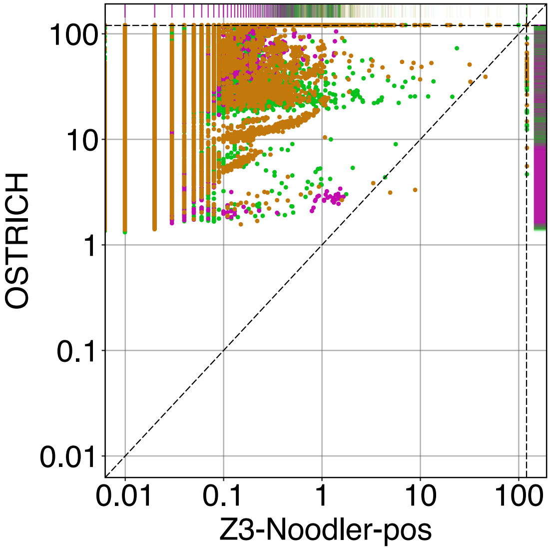
We conjecture that the problem (even when considering conjunctions) is NP-complete.

Conclusion

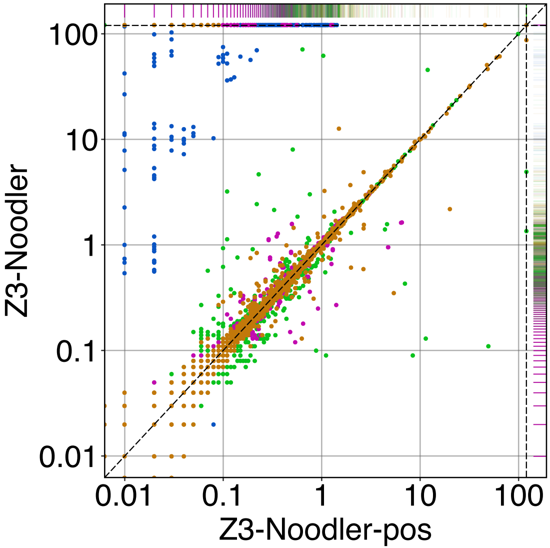
1. position constraints can be reduced to LIA using automata techniques
2. $\neg$*contains* is decidable

Thank you for you attention.
Questions?

Comparison with OSTRICH and Z3-NOODLER



(a) Z3-NOODLER-POS vs. OSTRICH



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