Z3-Noodler: Shepherding Decision Procedures

Vojtěch Havlena, et al.

Brno University of Technology, Czech Republic

MOSCA'25

SMT String constraint solving

- Checking **satisfiability** of formulae with **string variables** and operations

$$\underbrace{x = yz \land y \neq u}_{\text{(dis)equations}} \land \overbrace{x \in (ab)^* a^+ (b|c)}^{\text{regular constraints}} \land \overbrace{|x| = 2|u| + 1}^{\text{length constraints}} \land \underbrace{\text{contains}(u, \text{replace}(z, b, c)) \land \ldots}_{\text{more complex operations}}$$

SMT String constraint solving

- Checking **satisfiability** of formulae with **string variables** and operations

$$\underbrace{x = yz \land y \neq u}_{\text{(dis)equations}} \land \overbrace{x \in (ab)^* a^+ (b|c)}^{\text{regular constraints}} \land \overbrace{|x| = 2|u| + 1}^{\text{length constraints}} \land \underbrace{\mathsf{contains}(u, \mathsf{replace}(z, b, c)) \land \ldots}_{\text{more complex operations}}$$

- **Motivation**: **large and complex real-world programs** need security guarantees
  - **analysis** of string manipulating programs (**vulnerabilities** of web applications)

```
let x = y.substring(1, y.length - 1);
let z = y.concat(x);
assert(x === z);
```

$$x_0 = substr(y, 1, |y| - 1) \land$$
$$z_0 = y \cdot x_0 \land$$
$$x_0 \neq z_0$$

  - analysing **access control policies** (AWS/Rego)

```
action: deactivate,
resource: (a1, a2),
condition: {StringLike, s3:prefix, home*}
```

$$A = \text{"}deactivate\text{"} \land$$
$$(R = \text{"}a1\text{"} \lor R = \text{"}a2\text{"}) \land$$
$$prefix \in home^*$$

  - **verification** of cockpit systems (Boeing), etc.
  - $\rightsquigarrow$ **efficient and expressive** SMT string solvers are **needed**

Z3-Noodler: Highlight

- Based on SMT solver **Z3**
  - Z3-Noodler replaces Z3's **string theory solver**
  - low-level interaction with Z3 (given by the theory interface)
    - gathers conjunction of string constraints (provided by SMT core)
    - core Noodler's procedure
    - provide theory lemma
  - uses **Z3's linear arithmetic** (LIA) theory solver

Z3-Noodler: Highlight

- Based on SMT solver **Z3**
  - Z3-Noodler replaces Z3's **string theory solver**
  - low-level interaction with Z3 (given by the theory interface)
    - gathers conjunction of string constraints (provided by SMT core)
    - core Noodler's procedure
    - provide theory lemma
  - uses **Z3's linear arithmetic** (LIA) theory solver
- Uses **Nondeterministic finite automata** (**Mata** automata library) [TACAS'24]

Z3-Noodler: Highlight

- Based on SMT solver **Z3**
  - Z3-Noodler replaces Z3's **string theory solver**
  - low-level interaction with Z3 (given by the theory interface)
    - gathers conjunction of string constraints (provided by SMT core)
    - core Noodler's procedure
    - provide theory lemma
  - uses **Z3's linear arithmetic** (LIA) theory solver
- Uses **Nondeterministic finite automata** (**Mata** automata library) [TACAS'24]
- Combines various **decision procedures** for different fragments

Z3-Noodler: Highlight

- Based on SMT solver **Z3**
  - Z3-Noodler replaces Z3's **string theory solver**
  - low-level interaction with Z3 (given by the theory interface)
    - gathers conjunction of string constraints (provided by SMT core)
    - core Noodler's procedure
    - provide theory lemma
  - uses **Z3's linear arithmetic** (LIA) theory solver
- Uses **Nondeterministic finite automata** (**Mata** automata library) [TACAS'24]
- Combines various **decision procedures** for different fragments
- Support for various **string functions/predicates**
  - more **complex** string functions/predicates (`replace`, `substr`, `indexof` ...)
  - string-integer conversions, transducer constraints (`replace_all`)

Z3-Noodler: Highlight

- Based on SMT solver **Z3**
  - Z3-Noodler replaces Z3's **string theory solver**
  - low-level interaction with Z3 (given by the theory interface)
    - gathers conjunction of string constraints (provided by SMT core)
    - core Noodler's procedure
    - provide theory lemma
  - uses **Z3's linear arithmetic** (LIA) theory solver
- Uses **Nondeterministic finite automata** (**Mata** automata library) [TACAS'24]
- Combines various **decision procedures** for different fragments
- Support for various **string functions/predicates**
  - more **complex** string functions/predicates (`replace`, `substr`, `indexof` ...)
  - string-integer conversions, transducer constraints (`replace_all`)
- the **winner** of SMT-COMP'24 string division

Combining decision procedures

- **interface** of decision procedure
  - isSuitable(ψ) $\to \mathbb{B}$
  - init(ψ)
  - preprocess()
  - nextSolution() $\to \mathbb{B}_3$
  - getLIA() $\to \Phi_{\mathsf{LIA}} \times \{\mathsf{precise}, \mathsf{underapprox}\}$
  - getModel(θ, x) $\to \Sigma^*$

Combining decision procedures

- **interface** of decision procedure
  - isSuitable(ψ) $\rightarrow \mathbb{B}$
  - init(ψ)
  - preprocess()
  - nextSolution() $\rightarrow \mathbb{B}_3$
  - getLIA() $\rightarrow \Phi_{\text{LIA}} \times \{\text{precise}, \text{underapprox}\}$
  - getModel(θ, x) $\rightarrow \Sigma^*$
- procedure **selection**
  - suitability check
  - **ordered** by the most specific to the most general

Combining decision procedures

- **interface** of decision procedure
  - isSuitable(ψ) $\to \mathbb{B}$
  - init(ψ)
  - preprocess()
  - nextSolution() $\to \mathbb{B}_3$
  - getLIA() $\to \Phi_{\text{LIA}} \times \{\text{precise}, \text{underapprox}\}$
  - getModel(θ, x) $\to \Sigma^*$
- procedure **selection**
  - suitability check
  - **ordered** by the most specific to the most general
- **execution**
  - iteratively call nextSolution()
  - check if getLIA() is satisfiable

Stabilization-based procedure

[FM'23], [OOPSLA'23]

$$\underbrace{x = yz \quad \land \quad y \neq u}_{(dis)equations} \quad \land \quad \overbrace{x \in (ab)^*a^+(b|c)}^{regular\ constraints} \quad \land \quad \overbrace{|x| = 2|u| + 1}^{length\ constraints}$$

Stabilization-based procedure

[FM'23], [OOPSLA'23]

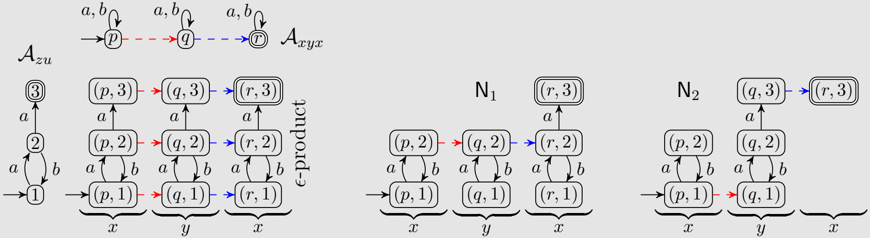
$$\underbrace{x = yz \quad \wedge \quad y \neq u}_{\text{(dis)equations}} \quad \wedge \quad \overbrace{x \in (ab)^* a^+ (b|c)}^{\text{regular constraints}} \quad \wedge \quad \overbrace{|x| = 2|u| + 1}^{\text{length constraints}}$$

- tight integration of **equations** with **regular constraints**
- complete on **chain-free constraints**
- iteratively **refining the languages** of variables *Lang*; until **stable form**
- feasible equation splits refining languages computed by **noodlification**
- create LIA formula **encoding possible lengths of words** in each language in *Lang*

Noodlification (No-lengths)

$xyx = zu$ | $u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^*$

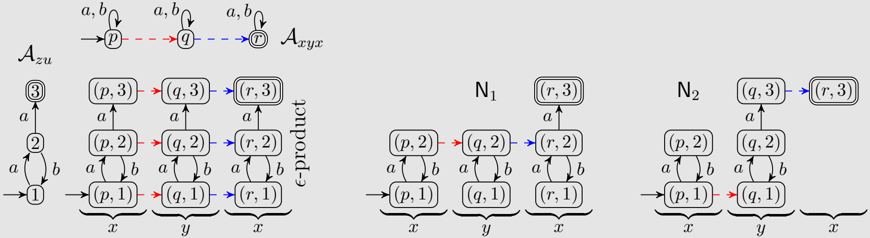
- Use right side to **refine languages** of variables x, y on the left side
- Leads to two **feasible noodles**:



Noodlification (No-lengths)

$\boldsymbol{xyx = zu}$ | $u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^*$

- Use right side to **refine languages** of variables x, y on the left side
- Leads to two **feasible noodles**:



- Two refinements

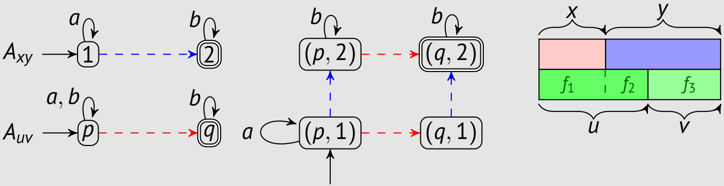
$u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto \boldsymbol{a} \quad y \mapsto \boldsymbol{(ba)^*}$

$u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto \boldsymbol{\epsilon} \quad y \mapsto \boldsymbol{a(ba)^*a}$

Noodlification (Lengths)

$xy = uv$ | $x \mapsto a^*$ $y \mapsto b^*$ $u \mapsto \Sigma^*$ $v \mapsto b^*$

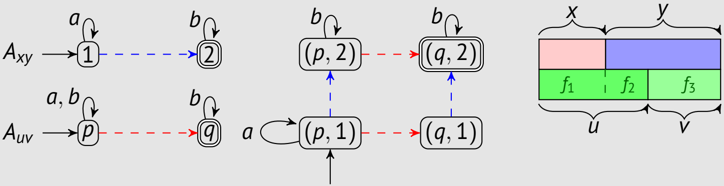
- **Align&Split** augmented with **noodlification** [Abdulla-CAV'14]
- **epsilon transitions** possibly on right side (only between relevant variables)
- epsilons defines **alignments** (derive substitutions with fresh variables)
- generate only **language-feasible splits**



Noodlification (Lengths)

$xy = uv$ | $x \mapsto a^*$ $y \mapsto b^*$ $u \mapsto \Sigma^*$ $v \mapsto b^*$

- **Align&Split** augmented with **noodlification** [Abdulla-CAV'14]
- **epsilon transitions** possibly on right side (only between relevant variables)
- epsilons defines **alignments** (derive substitutions with fresh variables)
- generate only **language-feasible splits**



$x = f_1$ $y = f_2 f_3$ $v = f_3$ $u = f_1 f_2$ | $x \mapsto a^*$ $y \mapsto b^*$ $u \mapsto \Sigma^*$ $v \mapsto b^*$ $f_1 \mapsto a^*$ $f_2 \mapsto b^*$ $f_3 \mapsto b^*$

$x = f_1$ $y = f_2$ $v = f_2$ $u = f_1$ | $x \mapsto a^*$ $y \mapsto b^*$ $u \mapsto \Sigma^*$ $v \mapsto b^*$ $f_1 \mapsto a^*$ $f_2 \mapsto b^*$

Example: $xx = w \wedge z = xy \wedge w \in a^+ \wedge |z| = 2|w| - |x|$

$xx = w \quad z = xy \;\mid\; w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\mid\;$

$\downarrow$ noodlify $(xx = w)$

$x = v_1 \quad w = v_1 v_1 \quad z = xy \;\mid\; v_1 \mapsto a^+ \quad w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\mid\;$

$\downarrow$ Subst $(x = v_1)$

$w = v_1 v_1 \quad z = v_1 y \;\mid\; v_1 \mapsto a^+ \quad w \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\mid\; x \mapsto v_1$

$\downarrow$ Subst $(w = v_1 v_1)$

$z = v_1 y \;\mid\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\mid\; x \mapsto v_1 \quad w \mapsto v_1 v_1$

$\downarrow$ Subst $(z = v_1 y)$

$\;\mid\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \;\mid\; x \mapsto v_1 \quad w \mapsto v_1 v_1 \quad z \mapsto v_1 y$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

$\boldsymbol{xx = w} \quad z = xy \;\big|\; \boldsymbol{w \mapsto a^+} \quad \boldsymbol{x \mapsto \Sigma^*} \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|$

$\downarrow$ **noodlify** $\quad \boldsymbol{(xx = w)}$

$\boldsymbol{x = v_1} \quad \boldsymbol{w = v_1 v_1} \quad z = xy \;\big|\; \boldsymbol{v_1 \mapsto a^+} \quad w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|$

$\downarrow$ Subst $\quad (x = v_1)$

$w = v_1 v_1 \quad z = v_1 y \;\big|\; v_1 \mapsto a^+ \quad w \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|\; x \mapsto v_1$

$\downarrow$ Subst $\quad (w = v_1 v_1)$

$z = v_1 y \;\big|\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|\; x \mapsto v_1 \quad w \mapsto v_1 v_1$

$\downarrow$ Subst $\quad (z = v_1 y)$

$\big|\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \;\big|\; x \mapsto v_1 \quad w \mapsto v_1 v_1 \quad z \mapsto v_1 y$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

$xx = w \quad z = xy$	$w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	

$\downarrow$ noodlify $(xx = w)$

$\boldsymbol{x = v_1} \quad w = v_1v_1 \quad z = xy$	$\boldsymbol{v_1 \mapsto a^+} \quad w \mapsto a^+ \quad \boldsymbol{x \mapsto \Sigma^*} \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	

$\downarrow$ **Subst** $(\boldsymbol{x = v_1})$

$w = v_1v_1 \quad z = v_1y$	$v_1 \mapsto a^+ \quad w \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	$\boldsymbol{x \mapsto v_1}$

$\downarrow$ Subst $(w = v_1v_1)$

$z = v_1y$	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1v_1$

$\downarrow$ Subst $(z = v_1y)$

	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1v_1 \quad z \mapsto v_1y$

Example: $xx = w \wedge z = xy \wedge w \in a^+ \wedge |z| = 2|w| - |x|$

$xx = w \quad z = xy \;\big|\; w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|\;$

$\downarrow$ noodlify $(xx = w)$

$x = v_1 \quad w = v_1v_1 \quad z = xy \;\big|\; v_1 \mapsto a^+ \quad w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|\;$

$\downarrow$ Subst $(x = v_1)$

$\mathbf{w = v_1v_1} \quad z = v_1y \;\big|\; \mathbf{v_1 \mapsto a^+} \quad \mathbf{w \mapsto a^+} \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|\; x \mapsto v_1$

$\downarrow$ **Subst** $(\mathbf{w = v_1v_1})$

$z = v_1y \;\big|\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\big|\; x \mapsto v_1 \quad \mathbf{w \mapsto v_1v_1}$

$\downarrow$ Subst $(z = v_1y)$

$\big|\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \;\big|\; x \mapsto v_1 \quad w \mapsto v_1v_1 \quad z \mapsto v_1y$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

$xx = w \quad z = xy$	$w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	

$\downarrow$ noodlify $(xx = w)$

$x = v_1 \quad w = v_1 v_1 \quad z = xy$	$v_1 \mapsto a^+ \quad w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	

$\downarrow$ Subst $(x = v_1)$

$w = v_1 v_1 \quad z = v_1 y$	$v_1 \mapsto a^+ \quad w \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*$	$x \mapsto v_1$

$\downarrow$ Subst $(w = v_1 v_1)$

$\boldsymbol{z = v_1 y}$	$\boldsymbol{v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^*}$	$x \mapsto v_1 \quad w \mapsto v_1 v_1$

$\downarrow$ **Subst** $\boldsymbol{(z = v_1 y)}$

	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1 v_1 \quad \boldsymbol{z \mapsto v_1 y}$

Example: $xx = w \wedge z = xy \wedge w \in a^+ \wedge |z| = 2|w| - |x|$

$xx = w \quad z = xy \quad | \quad w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \quad |$

$\downarrow$ noodlify $(xx = w)$

$x = v_1 \quad w = v_1 v_1 \quad z = xy \quad | \quad v_1 \mapsto a^+ \quad w \mapsto a^+ \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \quad |$

$\downarrow$ Subst $(x = v_1)$

$w = v_1 v_1 \quad z = v_1 y \quad | \quad v_1 \mapsto a^+ \quad w \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \quad | \quad x \mapsto v_1$

$\downarrow$ Subst $(w = v_1 v_1)$

$z = v_1 y \quad | \quad v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \quad | \quad x \mapsto v_1 \quad w \mapsto v_1 v_1$

$\downarrow$ Subst $(z = v_1 y)$

$| \quad v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad | \quad x \mapsto v_1 \quad w \mapsto v_1 v_1 \quad z \mapsto v_1 y$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+$	$y \mapsto \Sigma^*$	$x \mapsto v_1$	$w \mapsto v_1 v_1$	$z \mapsto v_1 y$

- stable solution ***(Lang, σ)***:
  - **language assignment** *Lang*: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1 v_1, z \mapsto v_1 y$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+$	$y \mapsto \Sigma^*$	$x \mapsto v_1$	$w \mapsto v_1 v_1$	$z \mapsto v_1 y$

- stable solution **$(Lang, \sigma)$**:
  - **language assignment** $Lang$: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1 v_1, z \mapsto v_1 y$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} \quad \land \quad \land \quad \land \quad \land$$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+$	$y \mapsto \Sigma^*$	$x \mapsto v_1$	$w \mapsto v_1 v_1$	$z \mapsto v_1 y$

- stable solution **$(Lang, \sigma)$**:
  - **language assignment** $Lang$: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1 v_1, z \mapsto v_1 y$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} |v_1| \geq 1 \land \qquad \land \qquad \land \qquad \land$$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1v_1 \quad z \mapsto v_1y$

- stable solution **$(Lang, \sigma)$**:
  - **language assignment** $Lang$: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1v_1, z \mapsto v_1y$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} |v_1| \geq 1 \land |y| \geq 0 \land \qquad \land \qquad \land$$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+$	$y \mapsto \Sigma^*$	$x \mapsto v_1$	$w \mapsto v_1 v_1$	$z \mapsto v_1 y$

- stable solution **(*Lang*, σ)**:
  - **language assignment** *Lang*: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1$, $w \mapsto v_1 v_1, z \mapsto v_1 y$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} |v_1| \geq 1 \land |y| \geq 0 \land |x| = |v_1| \land \qquad\qquad \land$$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1v_1 \quad z \mapsto v_1y$

- stable solution **$(Lang, \sigma)$**:
  - **language assignment** $Lang$: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, \boldsymbol{w \mapsto v_1v_1}, z \mapsto v_1y$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} |v_1| \geq 1 \land |y| \geq 0 \land |x| = |v_1| \land |w| = |v_1| + |v_1| \land$$

Example: $xx = w \wedge z = xy \wedge w \in a^+ \wedge |z| = 2|w| - |x|$

	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1v_1 \quad z \mapsto v_1y$

- stable solution ***(Lang, σ)***:
  - **language assignment** *Lang*: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1v_1, \boldsymbol{z \mapsto v_1y}$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} |v_1| \geq 1 \wedge |y| \geq 0 \wedge |x| = |v_1| \wedge |w| = |v_1| + |v_1| \wedge |z| = |v_1| + |y|$$

Example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

	$v_1 \mapsto a^+ \quad y \mapsto \Sigma^*$	$x \mapsto v_1 \quad w \mapsto v_1v_1 \quad z \mapsto v_1y$

- stable solution **$(Lang, \sigma)$**:
  - **language assignment** $Lang$: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1v_1, z \mapsto v_1y$
- **LIA formula** encoding **possible lengths** of variables:

$$\varphi_{\text{len}} \overset{\text{def.}}{\Leftrightarrow} |v_1| \geq 1 \land |y| \geq 0 \land |x| = |v_1| \land |w| = |v_1| + |v_1| \land |z| = |v_1| + |y|$$

- ask LIA solver if $|z| = 2|w| - |x| \land \varphi_{\text{len}}$ is satisfiable
  - it is, we have model $|v_1| = |x| = 1, |w| = |y| = 2, |z| = 3$
  - we can choose any word from $Lang(v_1)$ and $Lang(y)$ with correct lengths:
    $v_1 = a$ and $y = bc$
  - models for $x, w,$ and z are computed using the substitution map σ:
    $x = v_1 = a, w = v_1v_1 = aa,$ and $z = v_1y = abc$

Pure regular constraints

[TACAS'25]

$$\bigwedge_{1\leq i\leq n} x \in \mathcal{S}_i \quad \wedge \quad \bigwedge_{1\leq i\leq m} x \notin \mathcal{R}_i$$

- aut($\mathcal{S}$): **Regex $\rightsquigarrow$ NFA**
- $L(P) \cap L(U) \neq \emptyset$ where $P = \bigcap_{1\leq i\leq n} \mathrm{aut}(\mathcal{S}_i)$, $U = \bigcap_{1\leq i\leq m} \mathrm{aut}(\mathcal{R}_i)^\complement$
- **Problem**: **Expensive complement computation** (determinization) for negations

Pure regular constraints

[TACAS'25]

$$\bigwedge_{1\leq i\leq n} x \in \mathcal{S}_i \quad \wedge \quad \bigwedge_{1\leq i\leq m} x \notin \mathcal{R}_i$$

- aut($\mathcal{S}$): **Regex $\rightsquigarrow$ NFA**
- $L(P) \cap L(U) \neq \emptyset$ where $P = \bigcap_{1\leq i\leq n} \mathrm{aut}(\mathcal{S}_i)$, $U = \bigcap_{1\leq i\leq m} \mathrm{aut}(\mathcal{R}_i)^\complement$
- **Problem**: **Expensive complement computation** (determinization) for negations

- **eager simulation reduction/determinisation** (if necessary) for aut
- sort $\mathcal{S}_i$ by **estimated NFA size**
- **inclusion/universality checking**: $L(P) \not\subseteq L(V)$ where $V = \bigcup_{1\leq i\leq m} \mathrm{aut}(\mathcal{R}_i)$
  - **antichain-based algorithms**: perform well on real-world problems
- avoid construction for simpler cases

Pure regular constraints: Syntactic check

- Analyze **single regexes** ($x \in \mathcal{R}, x \notin \mathcal{R}$) to **extract syntactic properties**
- used to decide emptiness/universality
- **Propagate flags** (e, u, ℓ) through operations:
  - $e \in \mathbb{B}_3$: the regex includes the empty word $\qquad \mathbb{B}_3 = \{\top, \bot, \text{undef}\}$
  - $u \in \mathbb{B}_3$: the regex is universal
  - $\ell \in \mathbb{N} \cup \{\text{undef}\}$: the minimum length of words recognized by the regex

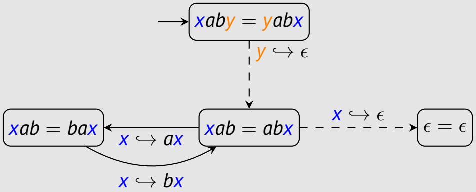
$$R_1 : (e_1, u_1, \ell_1) \qquad R_2 : (e_2, u_2, \ell_2) \qquad \texttt{re.++}(R_1, R_2)$$

$$(e_1 \wedge e_2, u, \ell_1 + \ell_2), \ell_1 + \ell_2 > 0 \rightsquigarrow u = \bot, \text{otherwise } u = u_1 \wedge u_2$$

- Completely **avoid the NFA construction by reasoning about the flags**
- undef: when flags are insufficient $\rightsquigarrow$ construct NFAs

Nielsen transformation

- **Quadratic** (each variable has at most two occurrences) and **beyond**
- Create a **Nielsen graph** (finite for a quadratic system)
  - node: set of equations, Nielsen tranformation metarules
  - priority queue; **pruning** of the state space
- lengths: **counter abstraction system** [LIN-LMCS'21]
  - heuristics for LIA formulae generation
  - iteratively generating LIA for **extended runs** (under-approximation)
  - **self-loop saturation**

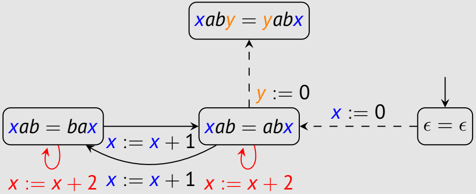


$$(x \hookrightarrow \alpha x) : \frac{(xu = \alpha v) \in \mathcal{E}}{\mathrm{trim}(\mathcal{E}[x/\alpha x])}$$

$$(x \hookrightarrow \epsilon) : \frac{(xu = v) \in \mathcal{E}}{\mathrm{trim}(\mathcal{E}[x/\epsilon])}$$

Nielsen transformation

- **Quadratic** (each variable has at most two occurrences) and **beyond**
- Create a **Nielsen graph** (finite for a quadratic system)
  - node: set of equations, Nielsen tranformation metarules
  - priority queue; **pruning** of the state space
- lengths: **counter abstraction system** [LIN-LMCS'21]
  - heuristics for LIA formulae generation
  - iteratively generating LIA for **extended runs** (under-approximation)
  - **self-loop saturation**



$$(x \hookrightarrow \alpha x) : \frac{(xu = \alpha v) \in \mathcal{E}}{\mathrm{trim}(\mathcal{E}[x/\alpha x])}$$

$$(x \hookrightarrow \epsilon) : \frac{(xu = v) \in \mathcal{E}}{\mathrm{trim}(\mathcal{E}[x/\epsilon])}$$

Nielsen transformation: Derivation of a LIA formula

$$xaby = yabx \land |x| \geq 50$$

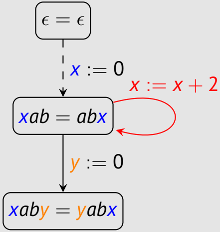
Nielsen transformation: Derivation of a LIA formula

$$xaby = yabx \land |x| \geq 50$$

$$\begin{tikzcd} \boxed{\epsilon = \epsilon} \arrow[d, dashed, "x := 0"] \\ \boxed{xab = abx} \arrow[loop right, "x := x + 2"] \arrow[d, "y := 0"] \\ \boxed{xaby = yabx} \end{tikzcd}$$

Nielsen transformation: Derivation of a LIA formula

$xaby = yabx \land |x| \geq 50$



- Fresh counter variables for each step

$$
\begin{aligned}
\varphi(x, y) \Leftrightarrow\ & x_0 = 0 \land y_0 = 0 \land \\
& x_1 = 0 \land y_1 = y_0 \land \\
& x_2 = x_1 + 2k \land y_2 = y_1 \land \\
& y_3 = 0 \land x_3 = x_2 \land \\
& x = x_3 \land y = y_3
\end{aligned}
$$

Nielsen transformation: Derivation of a LIA formula

$$xaby = yabx \land |x| \geq 50$$

$$\begin{tikzcd}
\boxed{\epsilon = \epsilon} \arrow[d, dashed, "x := 0"] \\
\boxed{xab = abx} \arrow[d, "y := 0"] \arrow[loop right, "x := x + 2"] \\
\boxed{xaby = yabx}
\end{tikzcd}$$

- Fresh counter variables for each step

$$\begin{aligned}
\varphi(x, y) \Leftrightarrow\ & x_0 = 0 \land y_0 = 0 \land \\
& x_1 = 0 \land y_1 = y_0 \land \\
& x_2 = x_1 + 2k \land y_2 = y_1 \land \\
& y_3 = 0 \land x_3 = x_2 \land \\
& x = x_3 \land y = y_3
\end{aligned}$$

Is $\varphi(|x|, |y|) \land |x| \geq 50$ satisfiable?

Length-based decision procedure

$$x = abyc \land x = zw \land x = uddc \land y = vad \land y = as \land |x| + |y| \geq 20$$

- Large systems (many equations, unrestricted variables and literals) $\rightsquigarrow$ **noodles explosion**
- positions of literals matters
- Symbolically **encode all possible alignments of literals** (their positions) into LIA formulae
- String formula reduced into a **LIA formula**

Length-based decision procedure

$$x = abyc \land x = zw \land x = uddc \land y = vad \land y = as \land |x| + |y| \geq 20$$

- Large systems (many equations, unrestricted variables and literals) $\rightsquigarrow$ **noodles explosion**
- positions of literals matters
- Symbolically **encode all possible alignments of literals** (their positions) into LIA formulae
- String formula reduced into a **LIA formula**

- **Block graph:** dependencies between blocks
- ***Block-acyclic* string constraint**: acyclic block graph

$$\bigwedge_{1 \leq i \leq n} x = R_i$$

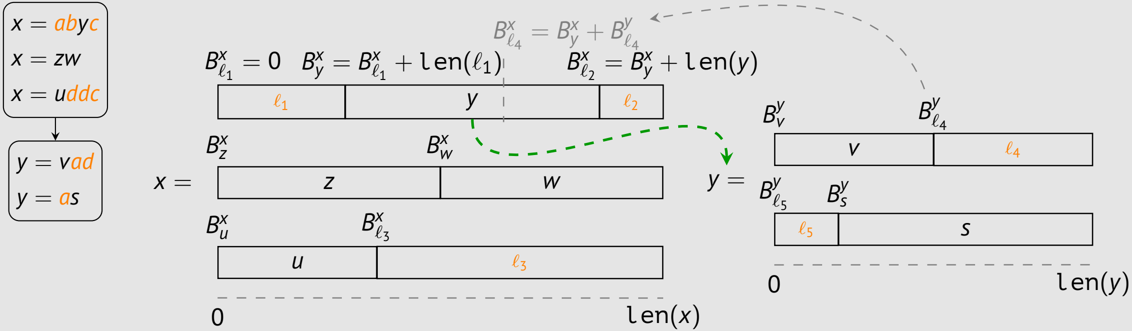
$$\boxed{\begin{array}{l} x = abyc \\ x = zw \\ x = uddc \end{array}}$$

$$\downarrow$$

$$\boxed{\begin{array}{l} y = vad \\ y = as \end{array}}$$

Length-based decision procedure: Alignments to LIA formula

$$x = abyc \wedge x = zw \wedge x = uddc \wedge y = vad \wedge y = as$$



Extended Constraints

- **string-integer conversions**: `to_int`, `from_code`, etc. [SAT'24]
  - stable solution (Lang, σ); set of conversion constraints $\mathcal{C} = \{k = \texttt{to\_int}(x), y = \texttt{from\_code}(l), \dots\}$, assume finite $\mathit{Lang}(x)$
  - create **concise** φ_c for $c \in \mathcal{C}$

Extended Constraints

- **string-integer conversions**: $\texttt{to\_int}$, $\texttt{from\_code}$, etc. [SAT'24]
  - stable **solution $(Lang, \sigma)$**; set of **conversion constraints** $\mathcal{C} = \{k = \texttt{to\_int}(x), y = \texttt{from\_code}(l), \dots\}$, assume finite $Lang(x)$
  - create **concise** φ_c for $c \in \mathcal{C}$
  - combining **lengths and conversions** $|x| \geq 3 \wedge \texttt{to\_int}(x) < 100$
  - handle **substitutions** $x \mapsto x_1 \cdots x_n \in \sigma$, ℓ_i: possible length of x_i (constant)

$$\texttt{to\_int}(x) = \sum_{1 \leq i \leq n} \left(\texttt{to\_int}(x_i) \cdot 10^{\ell_{i+1} + \dots + \ell_n}\right) \wedge \bigwedge_{1 \leq i \leq n} \left(|x_i| = \ell_i\right)$$

Extended Constraints

- **string-integer conversions**: `to_int`, `from_code`, etc. [SAT'24]
  - stable **solution** $(Lang, \sigma)$; set of **conversion constraints** $\mathcal{C} = \{k = \texttt{to\_int}(x), y = \texttt{from\_code}(l), \dots\}$, assume finite $Lang(x)$
  - create **concise** φ_c for $c \in \mathcal{C}$
  - combining **lengths and conversions** $|x| \geq 3 \land \texttt{to\_int}(x) < 100$
  - handle **substitutions** $x \mapsto x_1 \cdots x_n \in \sigma$, ℓ_i: possible length of x_i (constant)

$$\texttt{to\_int}(x) = \sum_{1 \leq i \leq n} \left(\texttt{to\_int}(x_i) \cdot 10^{\ell_{i+1} + \cdots + \ell_n}\right) \land \bigwedge_{1 \leq i \leq n} \left(|x_i| = \ell_i\right)$$

  - **intervals** $(0 \leq \texttt{to\_int}(x) \leq 7 \land |x| = 1) \lor (20 \leq \texttt{to\_int}(x) \leq 59 \land |x| = 2)$
  - encoding invalid strings
  - **underapproximation** of infinite languages

Extended Constraints

- **transducer constraints** $(T(xy, uv))$: `replace_all`
  - **transducer noodlification**: sequence of transducers
  - stable solution $\rightsquigarrow$ construct **multitape transducer** composing the same variables substitutions
  - Parikh image for lengths

Extended Constraints

- **transducer constraints** ($T(xy, uv)$): `replace_all`
  - **transducer noodlification**: sequence of transducers
  - stable solution $\rightsquigarrow$ construct **multitape transducer** composing the same variables substitutions
  - Parikh image for lengths
- **hard-positional constraints**: $\neg$`contains` [PLDI'25]
  - conversion to **quandified LIA** after stable solution found

Experimental evaluation

- **SMT-LIB benchmarks**, split into 3 categories:
  - **Regex** (mainly regular and length constraints): **AutomatArk**, **Denghang**, **Redos**, **StringFuzz**, **Sygus-qgen**
  - **Equations** (mostly word equations and length constraints with some small number of more complex constraints): **Kaluza**, **Kepler**, **Norn**, **Omark**, **Slent**, **Slog**, **Webapp**, **Woorpje**
  - **Predicates** (complex predicates): **FullStrInt**, **LeetCode**, **PyEx**, **StrSmallRw**, **Transducer+**
- Timeout: 120 s, memory limit: 8 GiB

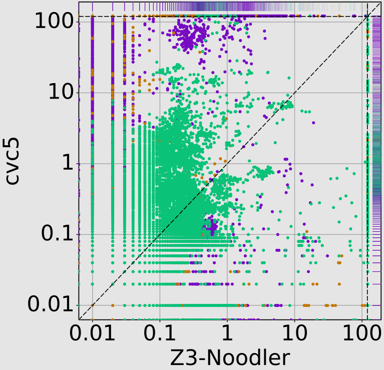
Experimental evaluation: Procedures comparison

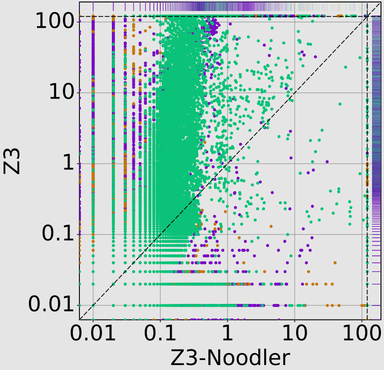
	number of calls	Regex proc.		Nielsen transf.		Length-based		Stabillization-based	
		called	solved	called	solved	called	solved	called	solved
Sygus-qgen	747	100%	100%	0%	0%	0%	0%	0%	0%
Denghang	999	0.10%	0.10%	0%	0%	96.10%	96.10%	3.80%	3.80%
AutomatArk	20,062	99.97%	99.97%	0%	0%	0.02%	0.02%	0.01%	0.01%
StringFuzz	9,941	46.45%	46.45%	0%	0%	27.98%	27.96%	25.58%	25.58%
Redos	2,952	70.02%	70.02%	0%	0%	11.21%	11.21%	18.77%	18.77%
Full **Regex**	34,701	79.21%	79.21%	0%	0%	11.75%	11.74%	9.04%	9.04%
LeetCode	874	1.37%	1.37%	0%	0%	59.27%	16.70%	81.92%	81.92%
StrSmallRw	6,327	0%	0%	0%	0%	4.85%	3.75%	96.25%	96.25%
PyEx	26,045	0.10%	0.10%	0%	0%	0.08%	0.08%	99.82%	99.82%
FullStrInt	9,003	0.04%	0.04%	0%	0%	0.26%	0.26%	99.70%	99.70%
Transducer+	0	-		-		-		-	
Full **Predicates**	42,249	0.10%	0.10%	0%	0%	2.06%	1.01%	98.89%	98.89%
Norn	918	11.76%	11.76%	0%	0%	6.86%	6.86%	81.37%	81.37%
Slog	1,565	25.37%	25.37%	0%	0%	0.13%	0.13%	74.50%	74.50%
Slent	1,489	0.40%	0.40%	0%	0%	35.19%	30.09%	69.51%	69.51%
Omark	9	0%	0%	11.11%	11.11%	11.11%	0%	88.89%	88.89%
Kepler	579	0%	0%	99.83%	99.83%	0%	0%	0%	0%
Woorpje	478	0.84%	0.84%	43.10%	42.47%	30.96%	27.20%	20.50%	20.50%
Webapp	381	0.52%	0.52%	0%	0%	2.36%	0.26%	99.21%	99.21%
Kaluza	11,222	35.31%	35.31%	0%	0%	63.45%	61.78%	2.91%	2.91%
Full **Equations**	16,641	26.92%	26.92%	4.72%	4.70%	47.27%	45.53%	22.59%	22.59%
All	93,591	34.20%	34.20%	0.84%	0.84%	13.69%	12.91%	52.01%	52.01%

Experimental evaluation: Running time

	Regex (32,242)		**Equations** (25,727)		**Predicates** (45,436)		All (103,405)	
	solved	time	solved	time	solved	time	solved	time
Z3-Noodler	32,232	3,688	25,301	1,147	45,035	6,353	**102,568**	**11,118**
Z3-Noodler$^{\mathcal{M}}$	32,228	4,010	25,299	1,456	45,035	7,321	**102,562**	**12,787**
cvc5	29,290	59,705	25,214	2,529	45,337	11,627	99,841	73,861
cvc5$^{\mathcal{M}}$	29,287	59,892	25,214	2,756	45,337	12,220	99,838	74,868
Z3	29,075	51,379	24,569	3,240	44,101	74,094	97,745	128,712
Z3$^{\mathcal{M}}$	29,064	51,830	24,571	4,013	44,096	74,708	97,731	130,551

Experimental evaluation: Comparison with other solvers





Times are in seconds, axes are logarithmic, timeouts on side dashed lines (120 s)

● **Regex**, ● **Equations**, and ● **Predicates**.

Conclusion and Acknowledgments

- **List of collaborators**:
  - František Blahoudek
  - Yu-Fang Chen
  - David Chocholatý
  - Vojtěch Havlena
  - Michal Hečko
  - Lukáš Holík
  - Jan Hranička
  - Ondřej Lengál
  - Juraj Síč

Conclusion and Acknowledgments

- **List of collaborators**:
  - František Blahoudek
  - Yu-Fang Chen
  - David Chocholatý
  - Vojtěch Havlena
  - Michal Hečko
  - Lukáš Holík
  - Jan Hranička
  - Ondřej Lengál
  - Juraj Síč

- **Future work**:
  - optimization of **transducer constraints**
  - application of Z3-Noodler on the **analysis of the security of web applications**