Cooking String-Integer Conversions with Noodles

or How to Extend String Solver Z3-Noodler with String-Integer Conversions

Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč

Brno University of Technology, Czech Republic

String constraint solving

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

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

- **Motivation**:
  - **analysis** of string manipulating programs (**vulnerabilities** of web applications, verification, etc.)

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

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

  - Amazon cloud **access control policies** [Rungta-CAV'22]

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

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

Our previous work

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

Our previous work

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

FM'23

- tight **integration** of eqs with reg. constr.
- works with **langs of variables**
- **refinement** and **noodlification**
- complete for **chain-free** fragment
- **prototype** implementation in Python

Our previous work

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

FM'23

- tight **integration** of eqs with reg. constr.
- works with **langs of variables**
- **refinement** and **noodlification**
- complete for **chain-free** fragment
- **prototype** implementation in Python

OOPSLA'23

- combines FM'23 with **Align&Split**
- length constr., diseqs, some complex op
- **linear-integer arithmetic** (LIA) encoding
- complete for **chain-free** fragment
- implemented in **Z3-Noodler**

This work

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

- Extends OOPSLA'23 procedure with handling **string-integer conversions**
  - `to_int`/`from_int` - string to/from integer:
    `to_int('0324')` $= 324$ `to_int('34a')` $= -1$ `from_int(134)` $=$ `'134'`
  - `to_code`/`from_code` - char to/from (Unicode) code point:
    `to_code('0')` $= 48$ `from_code(97)` $=$ `'a'` `to_code('ab')` $= -1$

Noodlification (FM'23) on an example

$$xyx = zu \land ww = xa \land u \in (baba)^*a \land z \in a(ba)^* \land x \in \Sigma^* \land y \in \Sigma^* \land w \in \Sigma^*$$

- $\Sigma = \{a, b\}$

Noodlification (FM'23) on an example

$$xyx = zu \land ww = xa \land u \in (baba)^*a \land z \in a(ba)^* \land x \in \Sigma^* \land y \in \Sigma^* \land w \in \Sigma^*$$

- $\Sigma = \{a, b\}$
- Regular constraints are **collected** in a **language assignment** represented by **automata**

$$Lang = \{u \mapsto (baba)^*a, z \mapsto a(ba)^*, x \mapsto \Sigma^*, y \mapsto \Sigma^*, w \mapsto \Sigma^*\}$$

Noodlification (FM'23) on an example

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

- $\Sigma = \{a, b\}$
- Regular constraints are **collected** in a **language assignment** represented by **automata**

$$Lang = \{u \mapsto (baba)^*a, z \mapsto a(ba)^*, x \mapsto \Sigma^*, y \mapsto \Sigma^*, w \mapsto \Sigma^*\}$$

Noodlification (FM'23) on an example

$\textcolor{darkred}{xyx = zu} \quad ww = xa \;|\; u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad w \mapsto \Sigma^*$

- $\Sigma = \{a, b\}$
- Regular constraints are **collected** in a **language assignment** represented by **automata**

$$Lang = \{u \mapsto (baba)^*a, z \mapsto a(ba)^*, x \mapsto \Sigma^*, y \mapsto \Sigma^*, w \mapsto \Sigma^*\}$$

- Use equations to **refine *Lang***, starting with **xyx = zu**
- For any solution (assignment v) string $s = \nu(x) \cdot \nu(y) \cdot \nu(x) = \nu(z) \cdot \nu(u)$ satisfies:

$$s \in \overbrace{\Sigma^*}^{x} \overbrace{\Sigma^*}^{y} \overbrace{\Sigma^*}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z} \overbrace{(baba)^*a}^{u}$$

- Use right side to **refine** languages of variables x, y on the left side by **noodlification**

Noodlification (FM'23) on an example

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

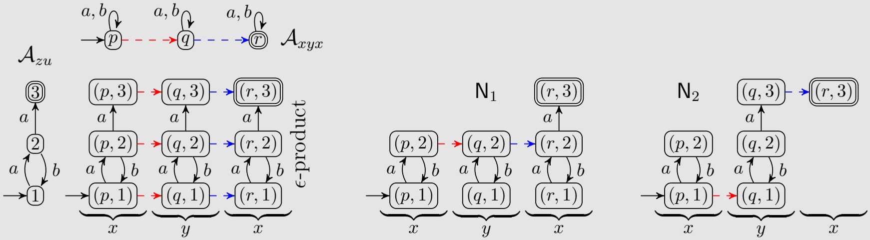
- Use right side to **refine** languages of variables x, y on the left side by **noodlification**

Noodlification (FM'23) on an example

$\mathbf{xyx = zu} \quad ww = xa \quad u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto \Sigma^* \quad y \mapsto \Sigma^* \quad w \mapsto \Sigma^*$

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

$$N_1 : \overbrace{\Sigma^*}^{x}\overbrace{\Sigma^*}^{y}\overbrace{\Sigma^*}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z}\overbrace{(baba)^*a}^{u} \qquad N_2 : \overbrace{\Sigma^*}^{x}\overbrace{\Sigma^*}^{y}\overbrace{\Sigma^*}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z}\overbrace{(baba)^*a}^{u}$$

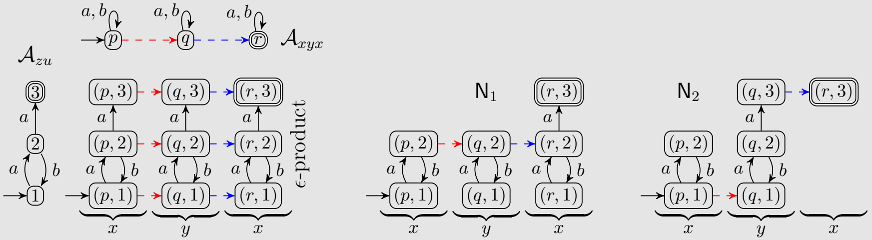


Noodlification (FM'23) on an example

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

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

$$N_1 : \overbrace{a}^{x}\ \overbrace{(ba)^*}^{y}\ \overbrace{a}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u} \qquad N_2 : \overbrace{\Sigma^*}^{x}\ \overbrace{\Sigma^*}^{y}\ \overbrace{\Sigma^*}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u}$$

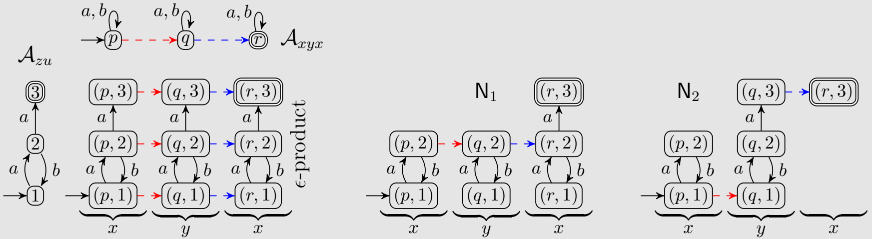


Noodlification (FM’23) on an example

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

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

$$N_1 : \overbrace{a}^{x}\ \overbrace{(ba)^*}^{y}\ \overbrace{a}^{x} \ \overset{=}{\cap}\ \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u} \qquad N_2 : \overbrace{\epsilon}^{x}\ \overbrace{a(ba)^*a}^{y}\ \overbrace{\epsilon}^{x} \ \overset{=}{\cap}\ \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u}$$

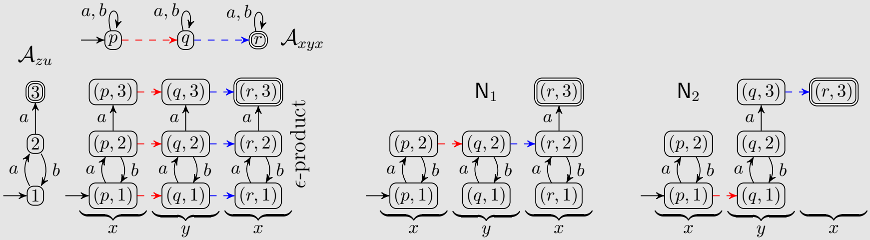


Noodlification (FM'23) on an example

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

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

$N_1 : \overbrace{a}^{x}\ \overbrace{(ba)^*}^{y}\ \overbrace{a}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u} \qquad N_2 : \overbrace{\epsilon}^{x}\ \overbrace{a(ba)^*a}^{y}\ \overbrace{\epsilon}^{x} \overset{=}{\cap} \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u}$



Noodlification (FM'23) on an example

$xyx = zu \quad \color{darkred}{ww = xa}$ | $u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto a \quad y \mapsto (ba)^* \quad w \mapsto \Sigma^*$

- Refine further with $\color{darkred}{ww = xa}$:

$$\overbrace{\Sigma^*}^{w}\overbrace{\Sigma^*}^{w} \overset{=}{\cap} \overbrace{a}^{x}\overset{a}{a}.$$

Noodlification (FM'23) on an example

$xyx = zu \quad ww = xa \;\big|\; u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto a \quad y \mapsto (ba)^* \quad w \mapsto a$

- Refine further with $ww = xa$:

$$\overbrace{a}^{w}\ \overbrace{a}^{w} \overset{=}{\cap} \overbrace{a}^{x}\ \overset{a}{a}.$$

Noodlification (FM'23) on an example

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

- Refine further with $ww = xa$:

$$\overbrace{a}^{w}\ \overbrace{a}^{w} \overset{=}{\cap} \overbrace{a}^{x}\ \overset{a}{a}.$$

- Languages in equations now **match**:

$$\overbrace{a}^{x}\ \overbrace{(ba)^*}^{y}\ \overbrace{a}^{x} \overset{=}{=} \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u} \quad \text{and} \quad \overbrace{a}^{w}\ \overbrace{a}^{w} \overset{=}{=} \overbrace{a}^{x}\ \overset{a}{a}.$$

Noodlification (FM'23) on an example

$$xyx = zu \quad ww = xa \quad u \mapsto (baba)^*a \quad z \mapsto a(ba)^* \quad x \mapsto a \quad y \mapsto (ba)^* \quad w \mapsto a$$

- Refine further with ***ww* = *xa***:

$$\overbrace{a}^{w}\ \overbrace{a}^{w}\ \overset{=}{\cap}\ \overbrace{a}^{x}\ \overset{a}{a}.$$

- Languages in equations now **match**:

$$\overbrace{a}^{x}\ \overbrace{(ba)^*}^{y}\ \overbrace{a}^{x}\ \overset{=}{=}\ \overbrace{a(ba)^*}^{z}\ \overbrace{(baba)^*a}^{u} \qquad \text{and} \qquad \overbrace{a}^{w}\ \overbrace{a}^{w}\ \overset{=}{=}\ \overbrace{a}^{x}\ \overset{a}{a}.$$

- *Lang* is a **stable solution** (we prove this is enough to decide it is SAT)

OOPSLA'23

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

- FM'23 can handle **equations** and **regular constraints** (at least **chain-free fragment**)
- How to handle more **complex operations** and **disequations**?
  - $\rightsquigarrow$ reduced (at least partially) to simpler constraints
- How to handle **lengths**?
  - create linear-integer arithmetic (LIA) formula **encoding possible lengths of words** in each language in *Lang*
  - stable solution *Lang* does not keep **dependencies** between lengths of vars
    - $\rightsquigarrow$ we use noodlification combined with **Align&Split** algorithm [Abdulla-CAV'14]

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

$\downarrow$ Align&Split $(xx = w)$

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

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

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

$\downarrow$ **Align&Split** $(\mathbf{xx = w})$

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

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

$$\boxed{w = v_1 v_1 \quad z = v_1 y \;\Big|\; v_1 \mapsto a^+ \quad w \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\Big|\; x \mapsto v_1}$$

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

$$\boxed{z = v_1 y \;\Big|\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \quad z \mapsto \Sigma^* \;\Big|\; x \mapsto v_1 \quad w \mapsto v_1 v_1}$$

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

$$\boxed{\;\Big|\; v_1 \mapsto a^+ \quad y \mapsto \Sigma^* \;\Big|\; x \mapsto v_1 \quad w \mapsto v_1 v_1 \quad z \mapsto v_1 y}$$

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

$\downarrow$ Align&Split $(xx = w)$

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

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

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

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

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

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

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

$$\Big\downarrow \ \text{Align\&Split} \quad (xx = w)$$

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

$$\Big\downarrow \ \text{Subst} \quad (x = v_1)$$

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

$$\Big\downarrow \ \textbf{Subst} \quad \mathbf{(w = v_1 v_1)}$$

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

$$\Big\downarrow \ \text{Subst} \quad (z = v_1 y)$$

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

$\downarrow$ Align&Split $(xx = w)$

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

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

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

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

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

$\downarrow$ **Subst** $(\mathbf{z = v_1y})$

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

$\downarrow$ Align&Split $(xx = w)$

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

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

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

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

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

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

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

OOPSLA'23 on 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, σ):
  - **language assignment** Lang: $v_1 \mapsto a^+, y \mapsto \Sigma^*$
  - **substitution map** σ: $x \mapsto v_1, w \mapsto v_1v_1, z \mapsto v_1y$

OOPSLA'23 on 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:

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

OOPSLA'23 on 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:

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

OOPSLA'23 on 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*, σ)**:
  - **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:

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

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

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

- 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:

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

OOPSLA'23 on example: $xx = w \wedge z = xy \wedge w \in a^+ \wedge |z| = 2|w| - |x|$

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

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

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

OOPSLA'23 on example: $xx = w \land z = xy \land w \in a^+ \land |z| = 2|w| - |x|$

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

- 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:

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

- 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:
    $x = v_1 = a, w = v_1 v_1 = aa,$ and $z = v_1 y = abc$

How to combine OOPSLA'23 with conversions?

- What we **have**:
  - stable **solution** $(Lang, \sigma)$
  - the **LIA part** of the initial formula $\mathcal{L}$
  - formula φ_{len} **encoding** possible **lengths** of variables
  - set of **conversion constraints** $\mathcal{C} = \{k = \mathtt{to\_int}(x), y = \mathtt{from\_code}(l), \ldots\}$
- How about **encoding** conversions into **LIA formula** too?
  - each conversion constraint $c \in \mathcal{C}$ encoded into **LIA formula** φ_c
  - $\varphi_{\mathsf{conv}} \overset{\text{def.}}{\Leftrightarrow} \bigwedge_{c \in \mathcal{C}} \varphi_c$
  - if $\mathcal{L} \wedge \varphi_{\mathsf{len}} \wedge \varphi_{\mathsf{conv}}$ is **satisfiable**, we have a **solution**
  - otherwise find **different** stable solution (if possible)

Handling $k = \texttt{to\_int}(x)$

- Semantics:
  - for a valid x (it contains only digits), k is the number represented by x
  - for an invalid x (it contains some non-digit), $k = -1$
- For stable solution $(Lang, \sigma)$ we have two distinct cases:
  - x is mapped to some language L_x in language assignment $Lang$
  - x is substituted by $x_1 \cdots x_n$ in substitution map σ

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in Lang$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying `to_int` on some word from L_x
- Generally possible only with **non-linear** arithmetic

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in Lang$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying `to_int` on some word from L_x
- Generally possible only with **non-linear** arithmetic
  $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)

Handling $k = \mathtt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in \mathit{Lang}$
- LIA formula $\varphi_{k=\mathtt{to\_int}(x)}$ should encode that k is the result of applying `to_int` on some word from L_x
- Generally possible only with **non-linear** arithmetic
  $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)
- We can iterate over all words:

$$\varphi_{k=\mathtt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} \bigvee_{w \in L_x} \left(\mathtt{to\_int}(x) = \mathtt{to\_int}(w)\right)$$

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in \mathit{Lang}$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying `to_int` on some word from L_x
- Generally possible only with **non-linear** arithmetic
  - $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)
- We can iterate over all words:

$$\varphi_{k=\texttt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} \bigvee_{w \in L_x} \left(\texttt{to\_int}(x) = \texttt{to\_int}(w)\right)$$

- Problems:

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in \mathit{Lang}$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying `to_int` on some word from L_x
- Generally possible only with **non-linear** arithmetic
  $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)
- We can iterate over all words:

$$\varphi_{k=\texttt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} \bigvee_{w \in L_x} \left(\texttt{to\_int}(x) = \texttt{to\_int}(w)\right)$$

- Problems:
  1. the **correspondence** between the length of x and the value of $\texttt{to\_int}(x)$

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in \mathit{Lang}$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying to_int on some word from L_x
- Generally possible only with **non-linear** arithmetic
  $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)
- We can iterate over all words:

$$\varphi_{k=\texttt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} \bigvee_{w \in L_x} \left(\texttt{to\_int}(x) = \texttt{to\_int}(w) \land |x| = |w|\right)$$

- Problems:
  1. the **correspondence** between the length of x and the value of $\texttt{to\_int}(x)$
     $\rightsquigarrow$ **relate** words with the corresponding length

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in Lang$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying $\texttt{to\_int}$ on some word from L_x
- Generally possible only with **non-linear** arithmetic
  $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)
- We can iterate over all words:

$$\varphi_{k=\texttt{to\_int}(x)} \stackrel{\text{def.}}{\Leftrightarrow} \bigvee_{w \in L_x} \left(\texttt{to\_int}(x) = \texttt{to\_int}(w) \land |x| = |w|\right)$$

- Problems:
  1. the **correspondence** between the length of x and the value of $\texttt{to\_int}(x)$
     $\rightsquigarrow$ **relate** words with the corresponding length
  2. can easily **blow-up**

Handling $k = \texttt{to\_int}(x)$ when x is in the language assignment

- Assume that $x \mapsto L_x \in \mathit{Lang}$
- LIA formula $\varphi_{k=\texttt{to\_int}(x)}$ should encode that k is the result of applying to_int on some word from L_x
- Generally possible only with **non-linear** arithmetic
  $\rightsquigarrow$ stronger restriction: L_x is **finite** (can be mitigated with **underapproximations**)
- We can iterate over all words:

$$\varphi_{k=\texttt{to\_int}(x)} \stackrel{\text{def.}}{\Leftrightarrow} \bigvee_{w \in L_x} \left(\texttt{to\_int}(x) = \texttt{to\_int}(w) \wedge |x| = |w|\right)$$

- Problems:
  1. the **correspondence** between the length of x and the value of $\texttt{to\_int}(x)$
     $\rightsquigarrow$ **relate** words with the corresponding length
  2. can easily **blow-up**
     $\rightsquigarrow$ encode **intervals** of words instead of single words

Intervals on an example

- Let $L_x = [0\text{-}7] \cup [2\text{-}5][0\text{-}9] \cup [3\text{-}6][0\text{-}9][0\text{-}9]$
- We create the following formula:

$$\varphi_{k=\mathtt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow}$$

Intervals on an example

- Let $L_x = \mathbf{[0\text{-}7]} \cup [2\text{-}5][0\text{-}9] \cup [3\text{-}6][0\text{-}9][0\text{-}9]$
- We create the following formula:

$$\varphi_{k=\mathtt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} (0 \leq \mathtt{to\_int}(x) \leq 7 \land |x| = 1)$$

Intervals on an example

- Let $L_x = [0\text{-}7] \cup \mathbf{[2\text{-}5][0\text{-}9]} \cup [3\text{-}6][0\text{-}9][0\text{-}9]$
- We create the following formula:

$$\varphi_{k=\mathtt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} (0 \leq \mathtt{to\_int}(x) \leq 7 \wedge |x| = 1) \\ \vee (20 \leq \mathtt{to\_int}(x) \leq 59 \wedge |x| = 2)$$

Intervals on an example

- Let $L_x = [0\text{-}7] \cup [2\text{-}5][0\text{-}9] \cup \mathbf{[3\text{-}6][0\text{-}9][0\text{-}9]}$
- We create the following formula:

$$\begin{aligned}\varphi_{k=\mathtt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} & (0 \leq \mathtt{to\_int}(x) \leq 7 \wedge |x| = 1) \\ & \vee (20 \leq \mathtt{to\_int}(x) \leq 59 \wedge |x| = 2) \\ & \vee (300 \leq \mathtt{to\_int}(x) \leq 699 \wedge |x| = 3)\end{aligned}$$

Intervals on an example

- Let $L_x = [0\text{-}7] \cup [2\text{-}5][0\text{-}9] \cup [3\text{-}6][0\text{-}9][0\text{-}9]$
- We create the following formula:

$$\begin{aligned}\varphi_{k=\mathtt{to\_int}(x)} \overset{\text{def.}}{\Leftrightarrow} & (0 \leq \mathtt{to\_int}(x) \leq 7 \wedge |x| = 1)\\ & \vee (20 \leq \mathtt{to\_int}(x) \leq 59 \wedge |x| = 2)\\ & \vee (300 \leq \mathtt{to\_int}(x) \leq 699 \wedge |x| = 3)\end{aligned}$$

- Easily implementable on automata level
- Handling invalid cases makes it a bit more complicated

Handling $k = \texttt{to\_int}(x)$ when x is in the substitution map

- Assume that $x \mapsto x_1 \cdots x_n \in \sigma$
- In stable solution, each x_i is mapped to some L_{x_i} in the language assignment *Lang*
- We can create LIA formulas encoding each $\texttt{to\_int}(x_i)$ using the interval method
- For each $(l_1, \ldots, l_n)$ with l_i some possible length of x_i we create

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

- $\varphi_{k = \texttt{to\_int}(x)}$ is defined as a disjunction of these equations
- Again, invalid cases make it more complicated

Handling $k = \texttt{to\_code}(x)$

- Semantics:
  - for a valid x (a char), k is the code points of x
  - for an invalid x (not a char), $k = -1$
- **Valid** part is always **finite**
  - **no problem** with **infinite** languages
  - we can iterate over all **characters**:

$$\varphi_{k=\texttt{to\_code}(x)} \stackrel{\text{def.}}{\Leftrightarrow} \bigvee_{a \in L_x \cap \Sigma} \texttt{to\_code}(x) = \texttt{to\_code}(a) \wedge |x| = 1$$

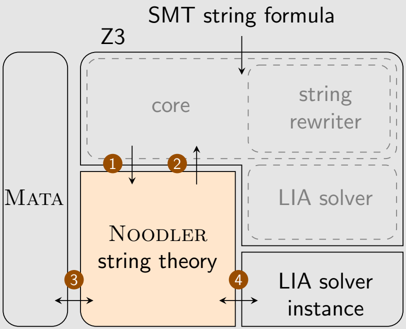
- Still problem with a **blow-up** (Σ is large)
  - set Σ_e of explicitly **used** symbols in formula is usually **small**
  - introduce a **special symbol** δ representing all **unused symbols**
  - work with a **much smaller** alphabet $\Sigma = \Sigma_e \cup \{\delta\}$
  - **special handling** of δ
- Needs to also encode the **correspondence** between $\texttt{to\_code}(x)$ and $\texttt{to\_int}(x)$

Handling `from_int/from_code`

- Very **similar** to `to_int/from_code`
- Instead of constraining the result, we want to constrain the argument
- We can use nearly the **same encoding**
- Slight **difference** in handling **invalid** cases

Implementation: Z3-Noodler

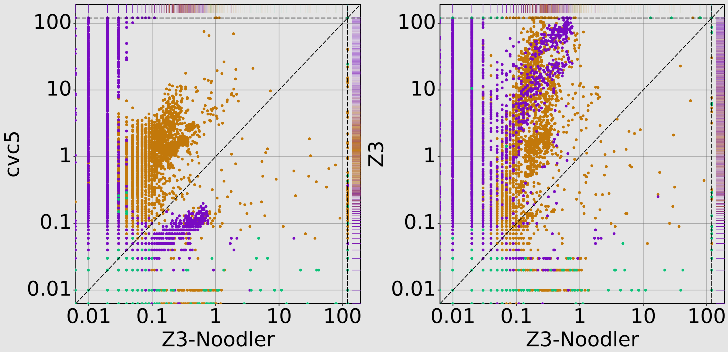
- Checks whether a given string constraint in **SMT** format is **satisfiable**
- Based on SMT solver **Z3**
  - formula **parsed** by Z3 and handled by **DPPL(T)**-based framework
  - Z3-Noodler replaces Z3's **string theory solver**
  - modified **string rewriter** (simplifications)
  - uses **Z3's linear arithmetic** (LIA) theory solver
- Uses **Mata**[1] library for handling finite automata
- **Winner** of SMT-COMP'24 string division



[1] Chocholatý, D. et al. Mata: A Fast and Simple Finite Automata Library. In: TACAS'24

Experiments

- SMT-LIB benchmarks containing conversions:
  - **FullStrInt**
  - **StringFuzz**
  - **StrSmallRw**
- Often **significantly faster** than other solvers



Conclusion

- We **extended** FM'23/OOPSLA'23 procedure with string-integer conversions
- Implemented in **Z3-Noodler**: `https://github.com/VeriFIT/z3-noodler`
- We can **beat** existing solvers
- Still complete on **chain-free** fragment (restriction of finite languages for `to_int`)
- What we are **currently** working on:
  - **model** generation (nearly done)
  - using **transducers** for `replace_all`
  - better handling of negated `contains`

Handling word disequations trough `to_code`

- In OOPSLA'23 we showed how to handle **arbitrary disequation** $s \neq t$:

$$\varphi_{s \neq t} \overset{\text{def.}}{\Leftrightarrow} |s| \neq |t| \vee \left(s = x_1 a_1 y_1 \wedge t = x_2 a_2 y_2 \wedge |x_1| = |x_2| \wedge a_1 \in \Sigma \wedge a_2 \in \Sigma \wedge \overbrace{a_1 \neq a_2}^{\text{dist}(a_1, a_2)} \right)$$

- Convoluted LIA formula $\text{dist}(a_1, a_2)$ computed after getting stable solution
- Important: this encoding has **no impact** on chain-free fragment
- Problem: encoding of $\text{dist}(a_1, a_2)$ is **incompatible** with conversions
- Solution:

$$\text{dist}(a_1, a_2) \overset{\text{def.}}{\Leftrightarrow} \texttt{to\_code}(a_1) \neq \texttt{to\_code}(a_2)$$

- Still **no impact** on chain-free fragment

Experiments

Table: Number of solved instances for each benchmark and tool, where *all* represents the full benchmark, while *conv* is a subset of formulae with at least one conversion constraint.

	FullStrInt		**StringFuzz**		**StrSmallRw**		**Σ**	
	all	conv	all	conv	all	conv	all	conv
total	16,968	16,130	11,618	1,608	1,880	80	30,466	17,818
Z3-Noodler	16,822	15,987	**11,616**	**1,606**	1,814	77	**30,252**	17,670
cvc5	**16,963**	**16,125**	10,915	1,579	**1,861**	**78**	29,739	**17,782**
Z3	16,729	15,896	11,081	1,565	1,821	**78**	29,631	17,539
OSTRICH	15,909	15,109	11,400	1,558	1,709	69	29,018	16,736
Z3-Noodler^{pr}	11,665	10,857	10,050	41	1,615	62	23,330	10,960