On Regular Constraint Propagation for Solving String Constraints

Philipp Rümmer
University of Regensburg
Uppsala University

EPFL, 2024-06-15

Joint work with
Artur Jeż, Matt Hague, Anthony W. Lin, Oliver Markgraf, and others

What are String Constraints?

Strings =

- finite sequences of letters over a finite alphabet (e.g., Unicode)

String contraints =

- quantifier-free formulas over string variables, including operations like:
  - String concatenation (word equations)
  - String length
  - Substring, character access
  - Regular expressions

Strings in Verification

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ε)
while(*){
      // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
      s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

Strings in Verification

ASCII, Unicode

// $Pre = (true)$
```
String s= '';
```
// $P_1 = (s \in \epsilon)$
```
while(*){
```
    // $P_2 = (s = u \cdot v \wedge u \in a^* \wedge v \in b^* \wedge |u| = |v|)$
```
    s= 'a' + s + 'b';
}
```
// $P_3 = P_2$
```
assert(!s.contains('ba') && (s.length() % 2) == 0);
```
// $Post = P_3$

Strings in Verification

Regular expression
assertion: $s = \epsilon$

```
// Pre = (      )
String s= '';
// P_1 = (s ∈ ε)
while(*){
    // P_2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P_3 = P_2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P_3
```

Strings in Verification

// $Pre = (true)$

`String s= '';`

// $P_1 = (s \in \epsilon)$

`while(*){`

    // $P_2 = (s = u \cdot v \wedge u \in a^* \wedge v \in b^* \wedge |u| = |v|)$

    `s= 'a' + s + 'b';`

`}`

// $P_3 = P_2$

`assert(!s.contains('ba') && (s.length() % 2) == 0);`

// $Post = P_3$

Word/string
concatenation

Strings in Verification

Loop invariant combining **word equations**, **regex** constraints, **length** constraints

```
// Pre = (true)
String s= '';
// P_1 = (s ∈ ε)
while(*){
    // P_2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P_3 = P_2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P_3
```

Strings in Verification

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ϵ)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

Substring constraint

Strings in Verification

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ε)
while(*){
      // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
      s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

Or **regex:**

$s \notin \Sigma^* \cdot ba \cdot \Sigma^*$

Strings in Verification

```
// Pre = (true)
String s= '';
// P_1 = (s ∈ ε)
while(*){
    // P_2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P_3 = P_2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P_3
```

Presburger length constraint

Strings in Verification

```
// Pre = (true)
String s= '';
// P1 = (s ∈ ε)
while(*){
    // P2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P3 = P2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P3
```

Strings in Verification

```
// Pre = (true)
String s= '';
// P_1 = (s ∈ ε)
while(*){
    // P_2 = (s = u · v ∧ u ∈ a* ∧ v ∈ b* ∧ |u| = |v|)
    s= 'a' + s + 'b';
}
// P_3 = P_2
assert(!s.contains('ba') && (s.length() % 2) == 0);
// Post = P_3
```

Main application area of string solving:
security analysis

AUTOMATED REASONING

A billion SMT queries a day

CAV keynote lecture by the director of applied science for AWS Identity explains how AWS is making the power of automated reasoning available to all customers.

By Neha Rungta

Share

August 18, 2022

At this year's Computer-Aided Verification (CAV) conference — a leading automated-reasoning conference collocated with the Federated Logic Conferences (FLoC) — Amazon's Neha Rungta delivered a keynote talk in which she suggested that innovations at Amazon have "ushered in the golden age of automated reasoning".

Amazon scientists and engineers are using automated reasoning to prove the correctness of critical internal systems and to help customers prove the security of their cloud infrastructures. Many of these innovations are being driven by powerful reasoning engines called SMT solvers.

A billion SMT queries a day

Neha Rungta's 2022 CAV keynote

Satisfiability problems, or SAT, ask whether it's possible to assign variables true/false values that satisfy a set of

Satisfiability Modulo Theories

- Many new SMT solvers for strings have emerged in the last years
- SMT-LIB theory of strings
- SMT-COMP has a `QF_Strings` division

Some String Solvers

- Hampi
- Kaluza
- Stranger
- Gecode+S
- GStrings
- Z3
- Z3-str/2/3/4/alpha
- CVC4/cvc5
- Norn
- S3/p/#
- TRAU
- nfa2sat
- OSTRICH
- Z3-Noodler
- Woorpje
- BEK, REX
- SLOG, SLENT
- *(many more)*

What's Decidable about ...

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

Word Equations

What's Decidable about ...

This is already in **solved form**

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

Word Equations

What's Decidable about ...

In general, two main decision procedures: **Makanin** & **Recompression**

This is already in **solved form**

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

Word Equations

What's Decidable about ...

In general, two main decision procedures: **Makanin** & **Recompression**

This is already in **solved form**

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

✓

Word Equations

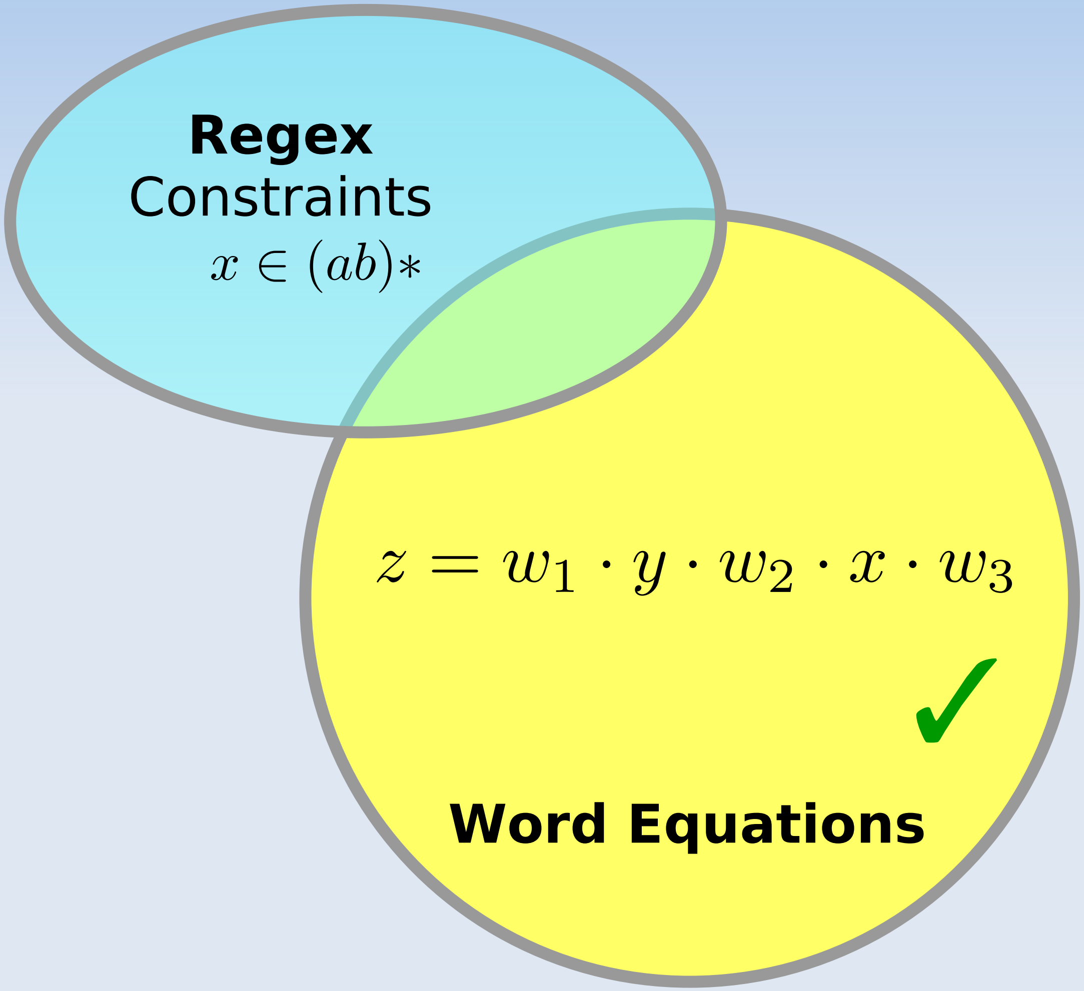
What's Decidable about ...

$$z = w_1 \cdot y \cdot w_2 \cdot x \cdot w_3$$

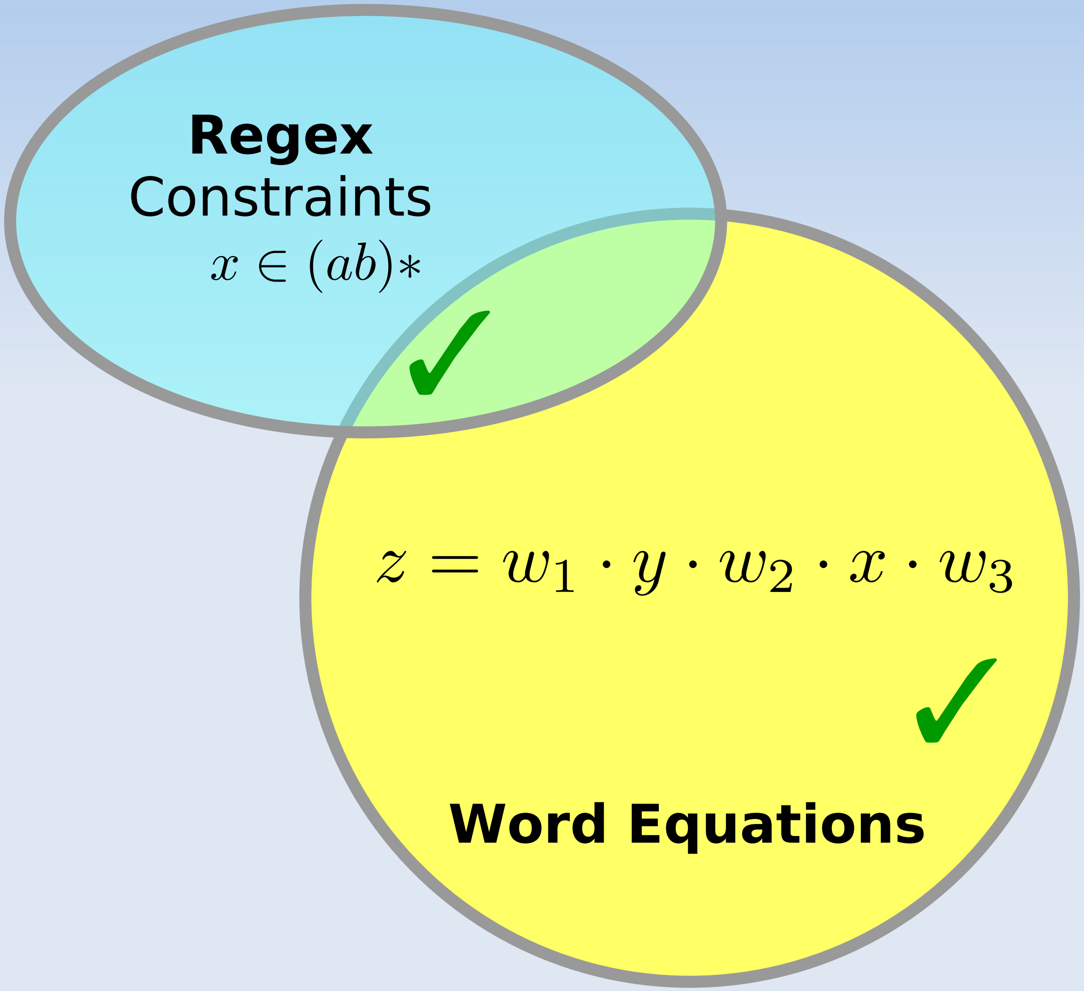
✓

Word Equations

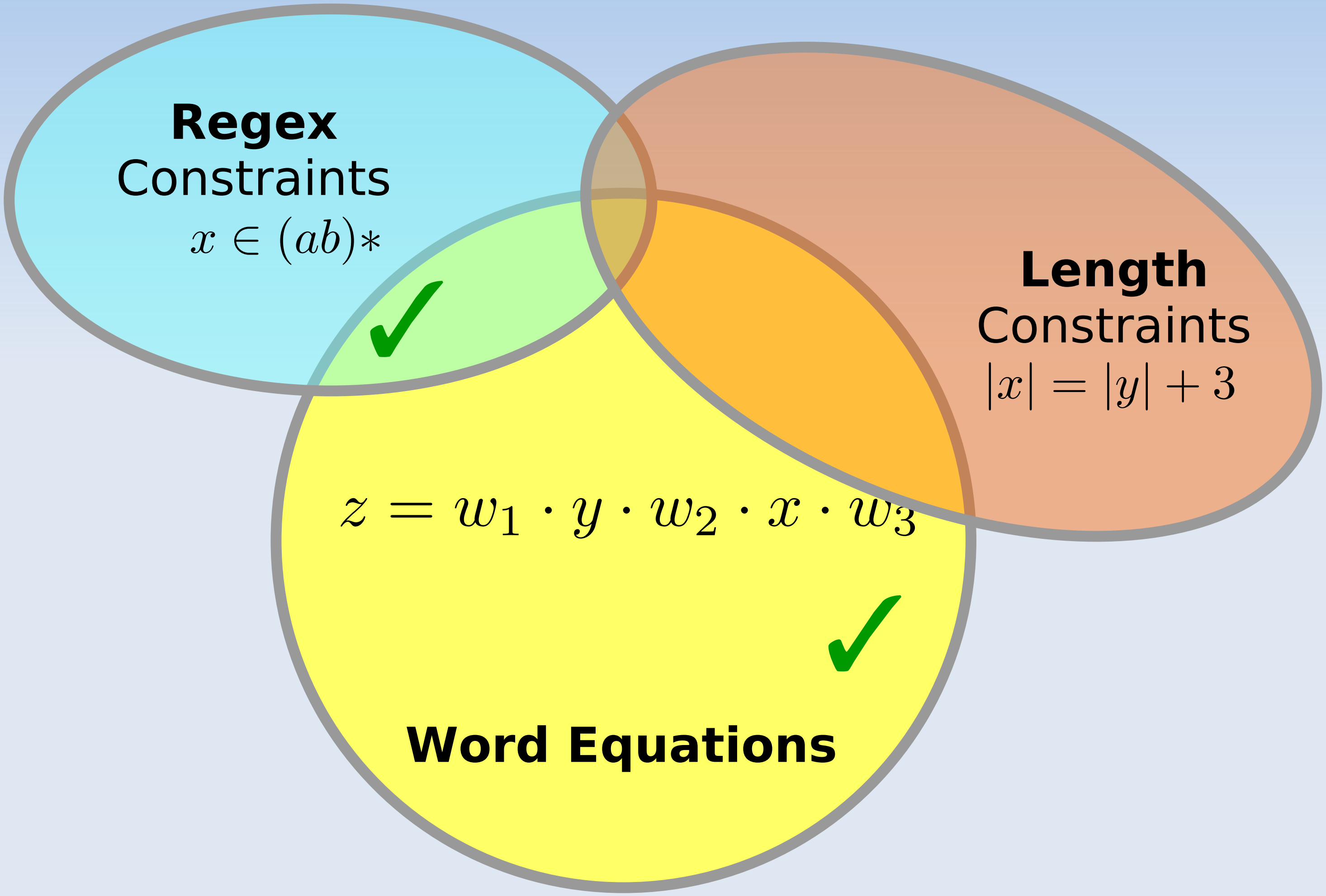
What's Decidable about ...



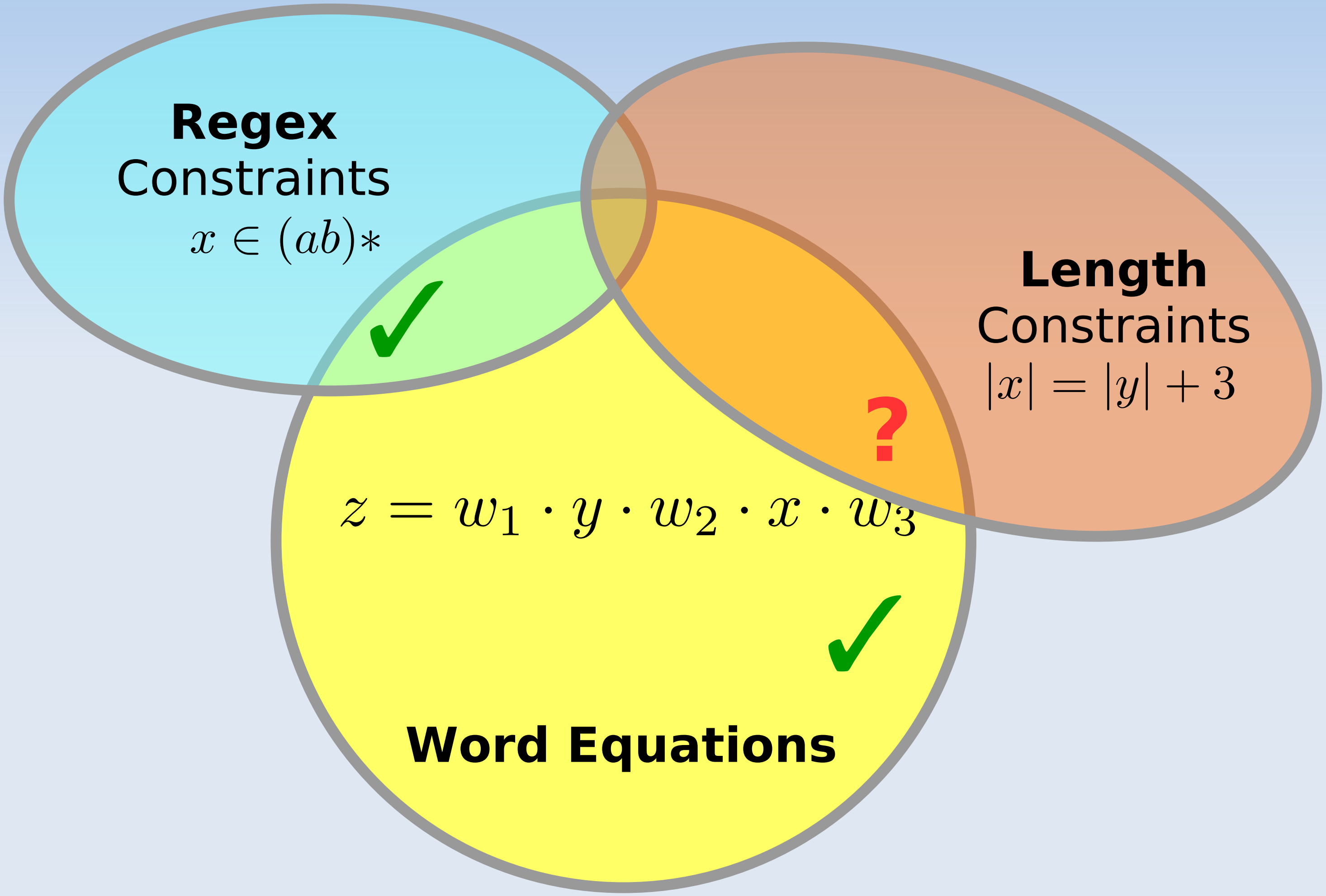
What's Decidable about ...



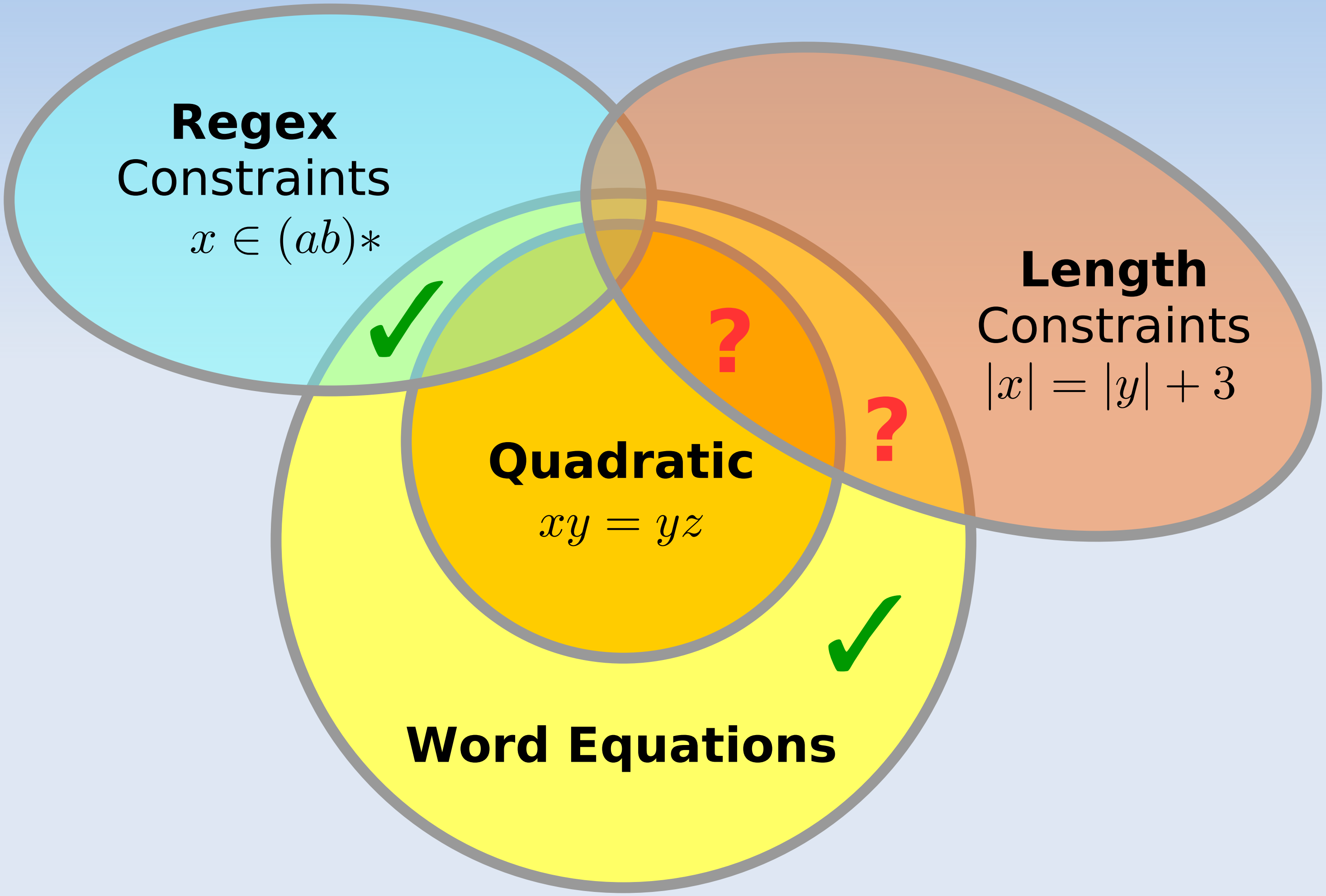
What's Decidable about ...



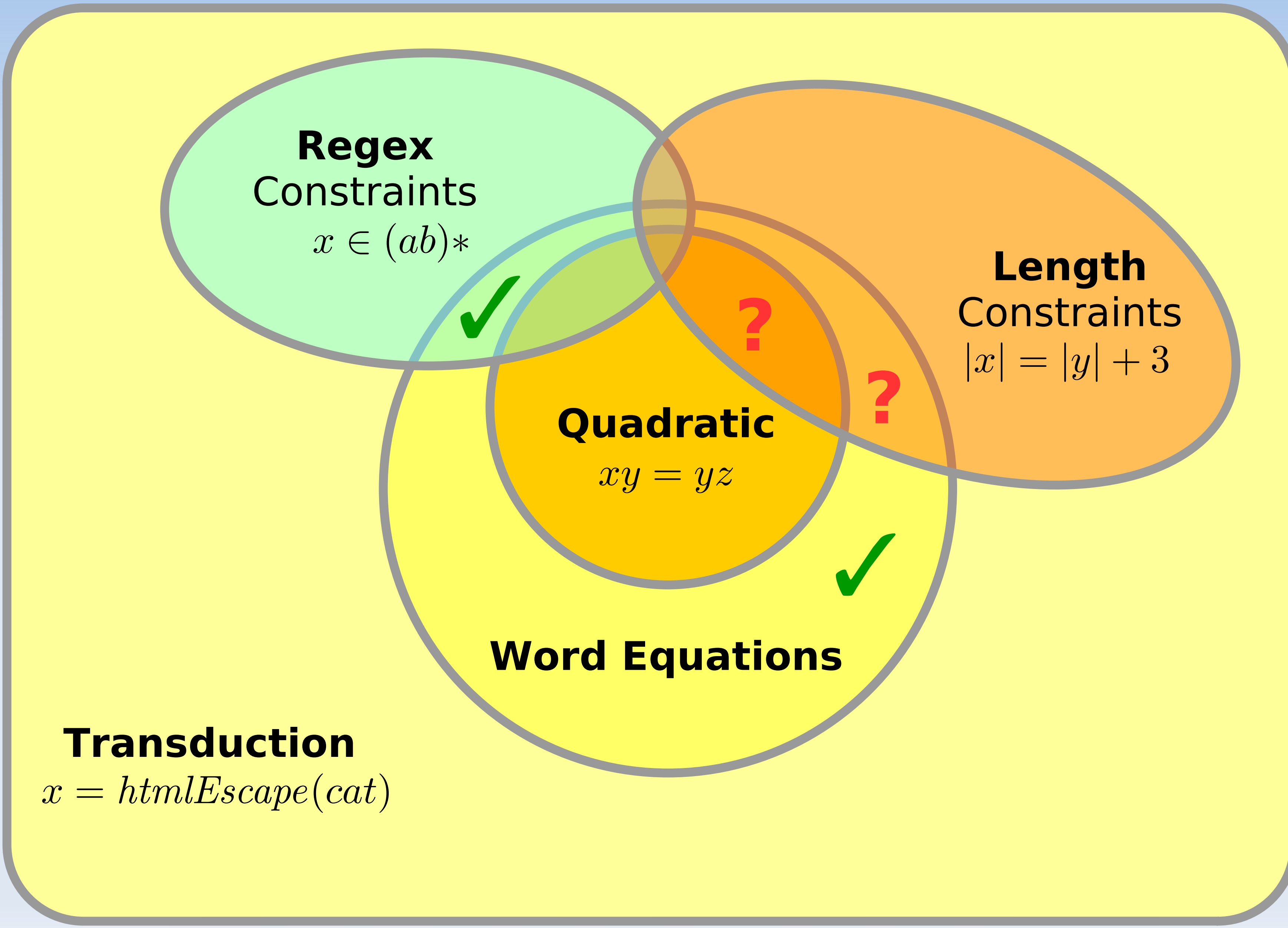
What's Decidable about ...



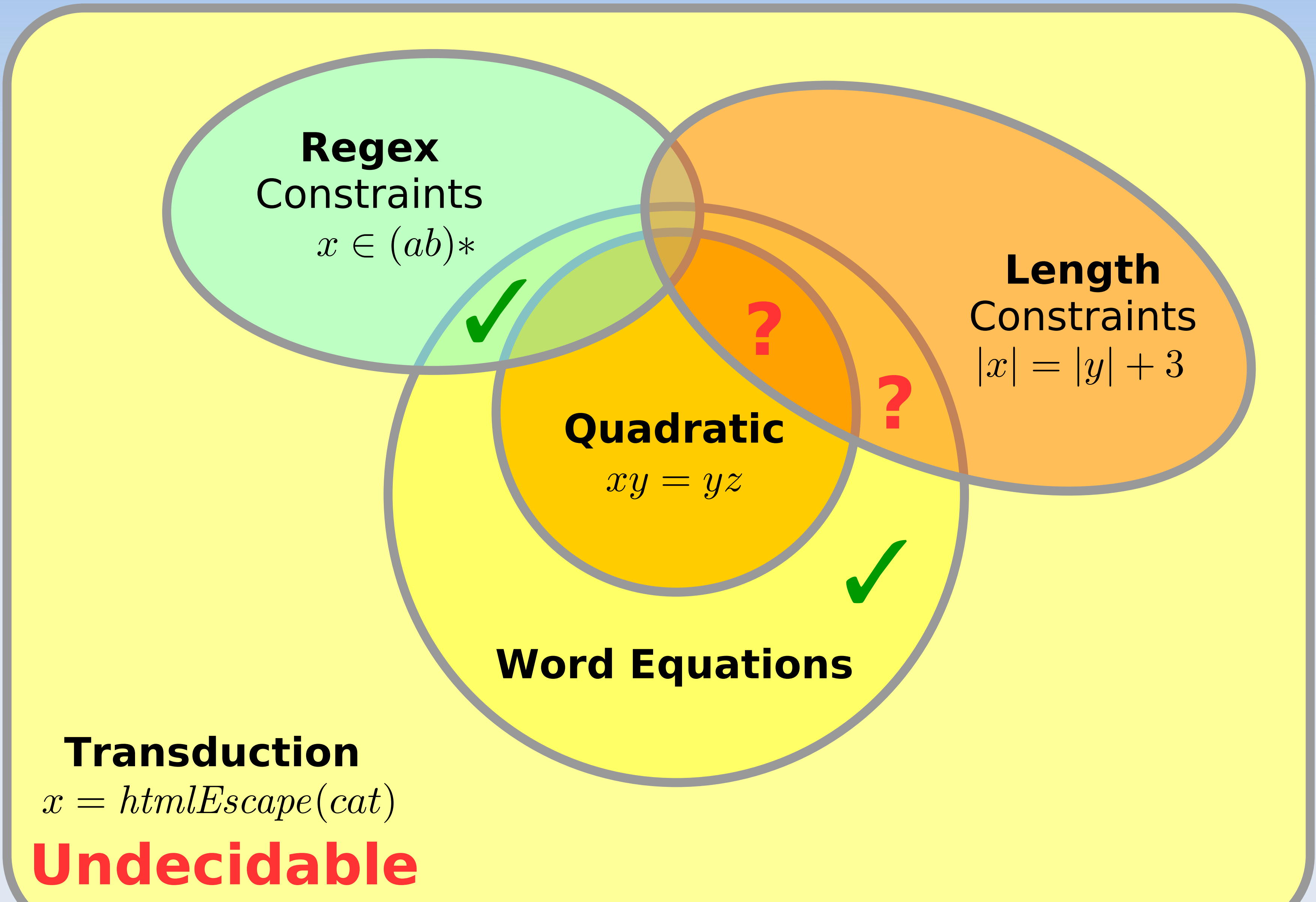
What's Decidable about ...



What's Decidable about ...



What's Decidable about ...



In Practice

Solvers use wide variety of techniques:

- Encoding as bit-vectors
- Encoding as SAT problem
- Rewriting/simplification rules
- Automata methods, derivatives
- Splitting rules for word equations
- (Re)Compression
- Propagation and CP methods
- *etc.*

Completeness

- Already for just word equations, decision procedures are **complicated** and **impractical** (not implementable?)
- String solvers are generally **incomplete** (for proving word equations unsat)

Completeness

- Already for just word equations, decision procedures are **complicated** and **impractical** (not implementable?)
- String solvers are generally **incomplete** (for proving word equations unsat)

- *Simpler decision procedures exist for various fragments (and are implemented by some solvers)*

Some Identified Fragments

- Quadratic word equations

- Tree-shaped
- Acyclic
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Some Identified Fragments

- Quadratic word equations

- Tree-shaped
- Acyclic
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Sometimes complicated definitions

Ad-hoc decision procedures

Relationship often unclear

Some Identified Fragments

- Quadratic word equations
- Tree shaped
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Is it possible to unify definitions?

Possible to have a **common proof format?**

Ad-hoc decision procedures

Relationship often unclear

Our Working Hypothesis

(Most/many/some/...) decision procedures can be understood as a combination of:

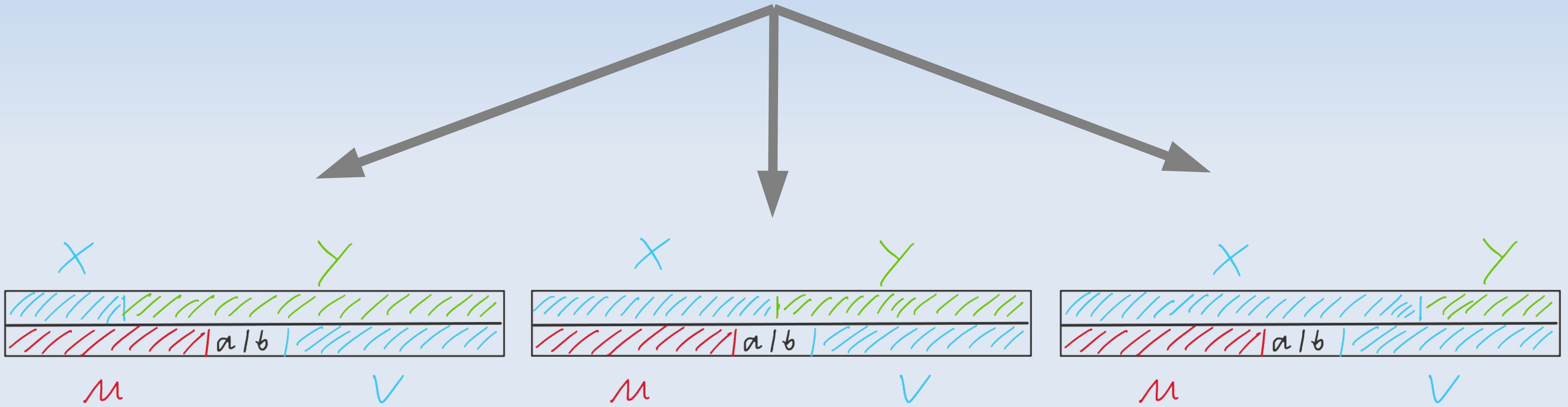
- Nielsen's transformation
- Propagation of regular constraints
- (Simplification rules)

Nielsen Transformation

$$x \cdot y = u \cdot \text{`}ab\text{'} \cdot v$$

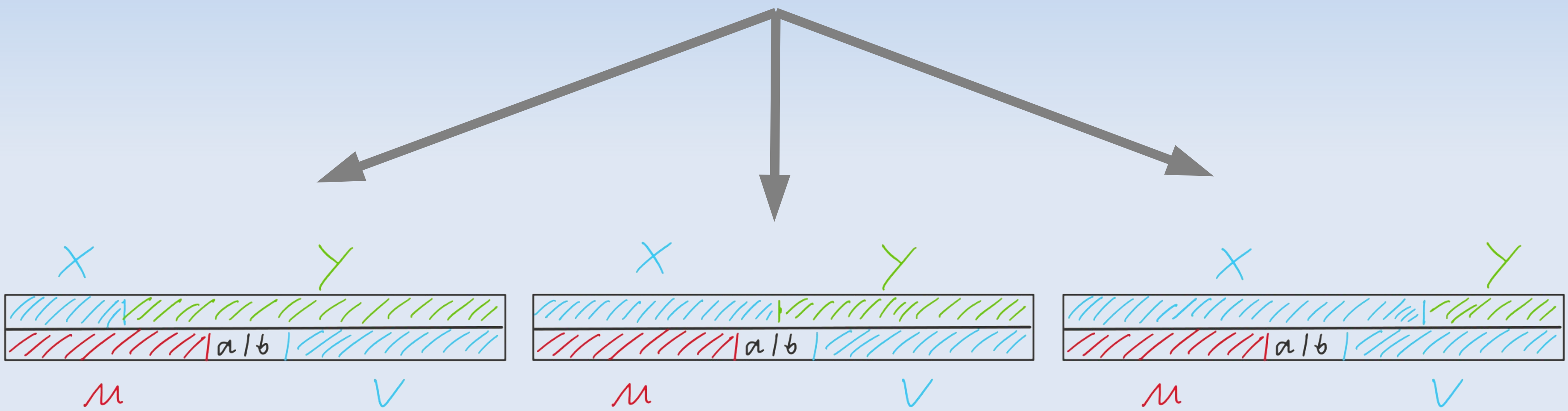
Nielsen Transformation

$$x \cdot y = u \cdot \text{`}ab\text{'} \cdot v$$



Nielsen Transformation

$$x \cdot y = u \cdot \text{`}ab\text{'} \cdot v$$



$$x = u_1 \wedge y = u_2 \cdot \text{`}ab\text{'} \cdot v \wedge u = u_1 \cdot u_2$$

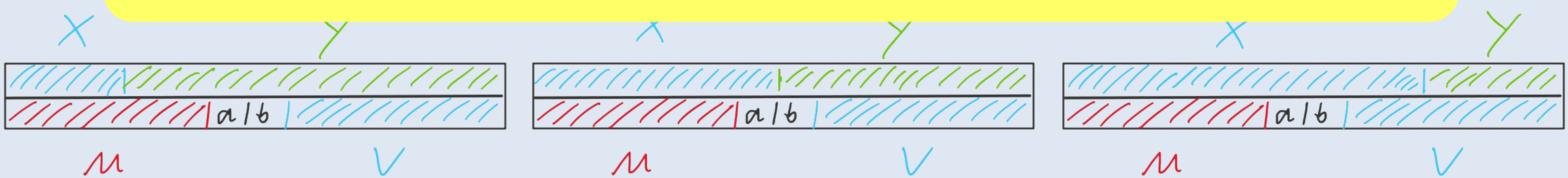
$$x = u \cdot \text{`}a\text{'} \wedge y = \text{`}b\text{'} \cdot v$$

$$x = u \cdot \text{`}ab\text{'} \cdot v_1 \wedge y = v_2 \wedge v = v_1 \cdot v_2$$

Nielsen Transformation

$$x \cdot y = u \cdot \text{`}ab\text{'} \cdot v$$

In general, this style of reasoning does not terminate. Most solvers split equations anyway.



$$\begin{aligned} & x = u_1 \\ \wedge\ & y = u_2 \cdot \text{`}ab\text{'} \cdot v \\ \wedge\ & u = u_1 \cdot u_2 \end{aligned}$$

$$\begin{aligned} & x = u \cdot \text{`}a\text{'} \\ \wedge\ & y = \text{`}b\text{'} \cdot v \end{aligned}$$

$$\begin{aligned} & x = u \cdot \text{`}ab\text{'} \cdot v_1 \\ \wedge\ & y = v_2 \\ \wedge\ & v = v_1 \cdot v_2 \end{aligned}$$

Our Working Hypothesis

(Most/many/some/...) decision procedures can be understood as a combination of:

- Nielsen's transformation
- Propagation of regular constraints
- (Simplification rules)

Constraint Normalization

String constraints can be rewritten to Boolean combinations of:

- Equations $x = y$
- Function applications $x = f(\bar{y})$
- Membership predicates $x \in \mathcal{L}$
- (ignoring length)

$x \cdot y =$ 'ab' becomes $z = concat(x, y) \wedge z \in ab$

Regular Constraint Prop.

$$\notin \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)} \text{ where } y \text{ is fresh} \qquad \textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$\text{=-}\textsc{Prop}\ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume}\ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2} \text{ if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$\text{=-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y} \text{ if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y} \text{ if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume}\ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect}\ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim}\ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop}\ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \qquad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

String Constraints with Real-World Regular Expressions. Taolue Chen, Matthew Hague, Zhilei Han, Denghang Hu, Alejandro Flores-Lamas, Anthony W. Lin, Shuanglong Kan, PR, Zhilin Wu, POPL'22

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\notin\ \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq\ \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)} \text{ where } y \text{ is fresh} \qquad \textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$=\text{-}\textsc{Prop}\ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume}\ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2} \text{ if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$=\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y} \text{ if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y} \text{ if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume}\ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect}\ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim}\ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop}\ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \qquad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

String Constraints with Real-World Regular Expressions. Taolue Chen, Matthew Hague, Zhilei Han, Denghang Hu, Alejandro Flores-Lamas, Anthony W. Lin, Shuanglong Kan, PR, Zhilin Wu, POPL'22

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \quad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\notin\ \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq\ \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)} \text{ where } y \text{ is fresh} \qquad \textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$=\text{-}\textsc{Prop}\ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume}\ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2} \text{ if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$=\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y} \text{ if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y} \text{ if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \quad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume}\ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \quad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect}\ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \quad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \quad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim}\ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \quad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop}\ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \quad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

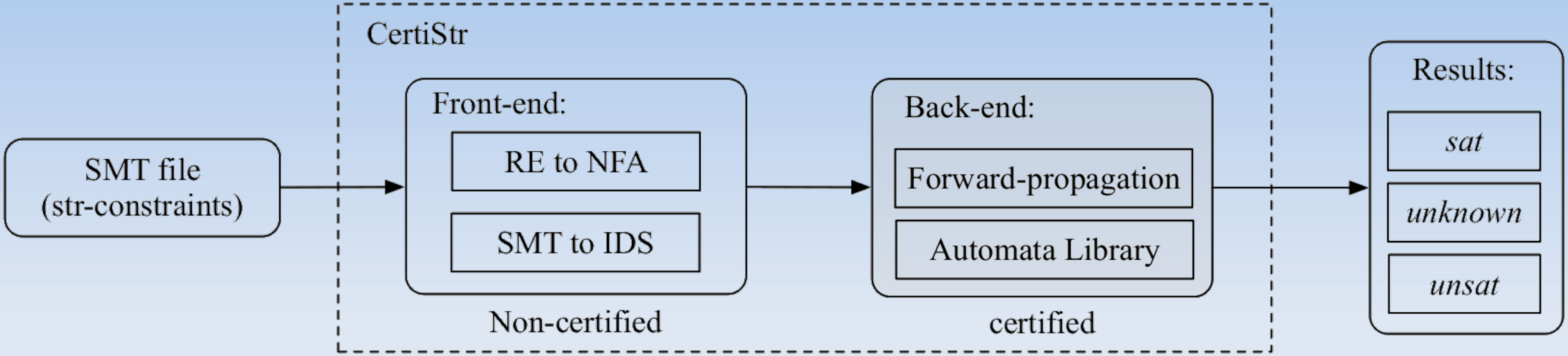
String Constraints with Real-World Regular Expressions. Taolue Chen, Matthew Hague, Zhilei Han, Denghang Hu, Alejandro Flores-Lamas, Anthony W. Lin, Shuanglong Kan, PR, Zhilin Wu, POPL'22

Forward Propagation

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \quad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

- "Image of regular languages is regular"
- Covers functions concat, transducers, etc.
- Over-approximate if argument variables not pairwise distinct
- Yields decision procedure for **tree-shaped constraints**

String Solver *CertiStr*



- Verified solver for tree-shaped constraints, written in Isabelle/HOL
- The hardest part: implementation of symbolic automata

CertiStr: a certified string solver. Shuanglong Kan, Anthony Widjaja Lin, PR, Micha Schrader, CPP'22

Regular Constraint Prop.

$$\notin\ \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq\ \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)} \text{ where } y \text{ is fresh} \qquad \textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$=\text{-}\textsc{Prop}\ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume}\ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2} \text{ if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$=\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y} \text{ if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y} \text{ if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume}\ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect}\ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim}\ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop}\ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \qquad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

String Constraints with Real-World Regular Expressions. Taolue Chen, Matthew Hague, Zhilei Han, Denghang Hu, Alejandro Flores-Lamas, Anthony W. Lin, Shuanglong Kan, PR, Zhilin Wu, POPL'22

$$\textsc{Bwd-Prop}\ \frac{\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \quad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

$$\notin\ \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq\ \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)} \ \text{where } y \text{ is fresh} \qquad \textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$=\text{-}\textsc{Prop}\ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume}\ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2} \ \text{if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$=\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y} \ \text{if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y} \ \text{if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume}\ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect}\ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim}\ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop}\ \frac{\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \quad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

Backward Propagation

$$\textsc{Bwd-Prop}\ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^{k}}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \quad \text{if} \quad \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^{k} \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

- “Pre-image of regular language is a recognizable relation”
- More general than forward prop.:
  - Also covers replace-all, etc.
  - Also precise when arguments coincide
- Yields decision procedure for **straightline constraints**

Some Fragments

- Quadratic word equations

- Tree-shaped
- Acyclic
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Some Fragments

- Quadratic word equations
- Tree-shaped
- Acyclic
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Forward + backward

Some Fragments

- Quadratic word equations
- Tree-shaped
- Acyclic
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Nielsen + backward, propagate cost-enriched regexes (aka Parikh automata)

Forward + backward

Some Fragments

- Quadratic word equations
- Tree-shaped
- Acyclic
- Chainfree
- Straightline
- Cost-enriched straightline
- Weakly chaining

Nielsen + backward, propagate cost-enriched regexes (aka Parikh automata)

Forward + backward

???

Some Fragments

???

- Quadratic word equations
- Tree-shaped
- Acyclic

Nielsen + backward,
propagate
cost-enriched regexes
(aka Parikh automata)

- Chainfree

Forward + backward

- Straightline
- Cost-enriched straightline
- Weakly chaining

???

Where is the limit?

Conjecture:

- Nielsen + forward + backward + cut is **incomplete** for both quadratic and general word equations

$$\notin \ \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq \ \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)} \ \text{where } y \text{ is fresh} \qquad \textsc{Cut} \ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$=\text{-}\textsc{Prop} \ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume} \ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2} \ \text{if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$=\text{-}\textsc{Prop-Elim} \ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y} \ \text{if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim} \ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y} \ \text{if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close} \ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume} \ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect} \ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop} \ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim} \ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop} \ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \qquad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

String Constraints with Real-World Regular Expressions. Taolue Chen, Matthew Hague, Zhilei Han, Denghang Hu, Alejandro Flores-Lamas, Anthony W. Lin, Shuanglong Kan, PR, Zhilin Wu, POPL'22

$$\textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$\notin\ \frac{\Gamma, x \in e^c}{\Gamma, x \notin e} \qquad \neq\ \frac{\Gamma, x \neq y, y = f(x_1, \ldots, x_n)}{\Gamma, x \neq f(x_1, \ldots, x_n)}\ \text{where } y \text{ is fresh}$$

$$\textsc{Cut}\ \frac{\Gamma, x \in e \qquad \Gamma, x \in e^c}{\Gamma}$$

$$=\text{-}\textsc{Prop}\ \frac{\Gamma, x \in e, x = y, y \in e}{\Gamma, x \in e, x = y} \qquad \neq\text{-}\textsc{Subsume}\ \frac{\Gamma, x \in e_1, y \in e_2}{\Gamma, x \in e_1, x \neq y, y \in e_2}\ \text{if } \mathcal{L}(e_1) \cap \mathcal{L}(e_2) = \emptyset$$

$$=\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e}{\Gamma, x \in e, x = y}\ \text{if } |\mathcal{L}(e)| = 1 \qquad \neq\text{-}\textsc{Prop-Elim}\ \frac{\Gamma, x \in e, y \in e^c}{\Gamma, x \in e, x \neq y}\ \text{if } |\mathcal{L}(e)| = 1$$

$$\textsc{Close}\ \frac{}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \emptyset$$

$$\textsc{Subsume}\ \frac{\Gamma, x \in e_1, \ldots, x \in e_n}{\Gamma, x \in e, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) \subseteq \mathcal{L}(e)$$

$$\textsc{Intersect}\ \frac{\Gamma, x \in e}{\Gamma, x \in e_1, \ldots, x \in e_n} \qquad \text{if } \begin{array}{l} n > 1 \text{ and} \\ \mathcal{L}(e_1) \cap \cdots \cap \mathcal{L}(e_n) = \mathcal{L}(e) \end{array}$$

$$\textsc{Fwd-Prop}\ \frac{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n))$$

$$\textsc{Fwd-Prop-Elim}\ \frac{\Gamma, x \in e, x_1 \in e_1, \ldots, x_n \in e_n}{\Gamma, x = f(x_1, \ldots, x_n), x_1 \in e_1, \ldots, x_n \in e_n} \qquad \text{if } \begin{array}{l} \mathcal{L}(e) = f(\mathcal{L}(e_1), \ldots, \mathcal{L}(e_n)) \\ \text{and } |\mathcal{L}(e)| = 1 \end{array}$$

$$\textsc{Bwd-Prop}\ \frac{\left\{\Gamma, x \in e, x = f(x_1, \ldots, x_n), x_1 \in e_1^i, \ldots, x_n \in e_n^i\right\}_{i=1}^k}{\Gamma, x \in e, x = f(x_1, \ldots, x_n)} \qquad \text{if } \begin{array}{l} f^{-1}(\mathcal{L}(e)) = \\ \bigcup_{i=1}^k \left(\mathcal{L}(e_1^i) \times \cdots \times \mathcal{L}(e_n^i)\right) \end{array}$$

String Constraints with Real-World Regular Expressions. Taolue Chen, Matthew Hague, Zhilei Han, Denghang Hu, Alejandro Flores-Lamas, Anthony W. Lin, Shuanglong Kan, PR, Zhilin Wu, POPL'22

The Next Steps

- **Properties** and **generalization** of regular constraint propagation
- What's **decidable** about strings constaints?
- From certified solver to proof checker: **proof format** for string solvers

For more details: see our POPL'24 tutorial
https://eldarica.org/ostrich-popl24/

Towards Alethe-Style Proofs

```
(regular-languages
  (! (re.from_automaton "automaton {init s0; ...;") :id 1)
  (! (re.from_automaton "automaton {init s0; ...;") :id 2)
  (! (re.from_automaton "automaton {init s0; ...;") :id 3)
 )

(assume h0 (str.in_re_id w 1))
(assume h1 (or (str.in_re_id w 2) (str.in_re_id w 3)))
(step t2 (cl (str.in_re_id w 2) (str.in_re_id w 3)) :rule or :premises (h1))

; start proof branch that spans until t3
(anchor :step t3)
(assume t3.h0 (str.in_re_id w 2))

(step t3.t1 (cl (not (str.in_re_id w 1)) (not (str.in_re_id w 2)))
                                             :rule re_empty_intersection)
(step t3.t2 (cl) :rule resolution :premises (h0 t3.h0 t3.t1))

(subproof t3 (cl (not (str.in_re_id w 2)))

(step t4 (cl (str.in_re_id w 3)) :rule resolution :premises (t2 t3))

(step t5 (cl (not (str.in_re_id w 1)) (not (str.in_re_id w 3)))
                                          :rule re_empty_intersection)
(step t6 (cl) :rule resolution :premises (h0 h2 t5))
```

Straightline Example

x in a*c*b*

y := reverse(x)

y in b*a*

z := replaceAll(y, a, b)

z in b*

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

y := reverse(x)

y in b*a*

$$\left.\begin{array}{l}\text{z := replaceAll(y, a, b)}\\ \text{z in b*}\end{array}\right\}\ \text{y in (a | b)*}$$

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

y := reverse(x)

y in b*a*

y in (a | b)*

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

y := reverse(x)

y in b*a*

y in (a | b)*

y in (a | b)* & b*a*

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

y := reverse(x)

y in b*a*

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

y := reverse(x)

y in b*a*

} x in a*b*

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

x in a*b*

Is there an input x satisfying all assertions?

Straightline Example

x in a*c*b*

x in a*b*

} x in a*b*

Is there an input x satisfying all assertions?

Straightline Example

x in a*b*

Is there an input x satisfying all assertions?

Straightline Example

Easy to solve!

x in a*b*

Is there an input x satisfying all assertions?

Straightline Example

Easy to solve!

x in a*b*

Solution: x = abb