A DPLL(T) Theory Solver for Strings and Regular Expressions

Tianyi Liang
Andrew Reynolds
Cesare Tinelli
University of Iowa
Morgan Deters
Clark Barrett
New York University

Motivation : Security Applications

```cpp
char buff[15];
char pass;
cout << "Enter the password :";
gets(buff);
if (regex_match(buff, std::regex("([A-Z]+)") )) {
    if(strcmp(buff, "PASSWORD")) {
        cout << "Wrong Password";
    } else {
        cout << "Correct Password";
        pass = 'Y';
    }
    if(pass == 'Y') {
        /* Grant the root permission*/
    }
}
```

Encode

```
(set-logic QF_S)
(declare-const input String)
(declare-const buff String)
(declare-const pass0 String)
(declare-const rest String)
(declare-const pass1 String)
(assert (= (str.len buff) 15))
(assert (= (str.len pass1) 1))
(assert (or (< (str.len input) 15)
    (= input (str.++ buff pass0 rest))))
(assert (str.in.re buff
            (re.+ (re.range "A" "Z"))))
(assert (ite (= buff "PASSWORD")
             (= pass1 "Y")
             (= pass1 pass0)))
(assert (not (= buff "PASSWORD")))
(assert (= pass1 "Y"))
```

Solve

Extract

```
tiliang@milner:~/workspace/security/benchmarks/homemade$ ~/CVC4/bin/pt-cvc4 propsalex.smt2
sat
(model
(define-fun input () String "AAAAAAAAAAAAAAAY")
(define-fun buff () String "AAAAAAAAAAAAAAA")
(define-fun pass0 () String "Y")
(define-fun rest () String "")
(define-fun pass1 () String "Y")
)
```

Objectives

- Want solver to handle:
  - (Unbounded) string constraints
  - Length constraints
  - Regular language memberships, …

Focus of this talk

- Theoretical complexity of:
  - Word equation problem: PSPACE
  - …with length constraints: OPEN
  - …with other functions (e.g. `replace`): UNDECIDABLE

Objectives

- Instead, focus on solver that is:
  - Efficient in practice
  - Tightly integrated into SMT architecture
    - Conflict analysis, T-propagation, lemma learning, combination of theories, …
  - Robust

Core Language for Theory of Strings

- Terms are:
  - Constants from a fixed finite alphabet Σ^* (a, ab, cbc...)
  - Free constants or "variables" (x, y, z, w...)
  - String concatenation

    $\_ \cdot \_$: String $\times$ String $\rightarrow$ String
  - Length terms

    `len(_)` : String $\rightarrow$ Int
- Example input:

$$\texttt{len(x)>len(y)} \wedge (\texttt{x}\cdot\texttt{b=y}\cdot\texttt{ab} \vee \texttt{x=y})$$

DPLL(T): Find Satisfying Assignment

```
len(x)>len(y)
(x·b=y·ab ∨ x=y)
```

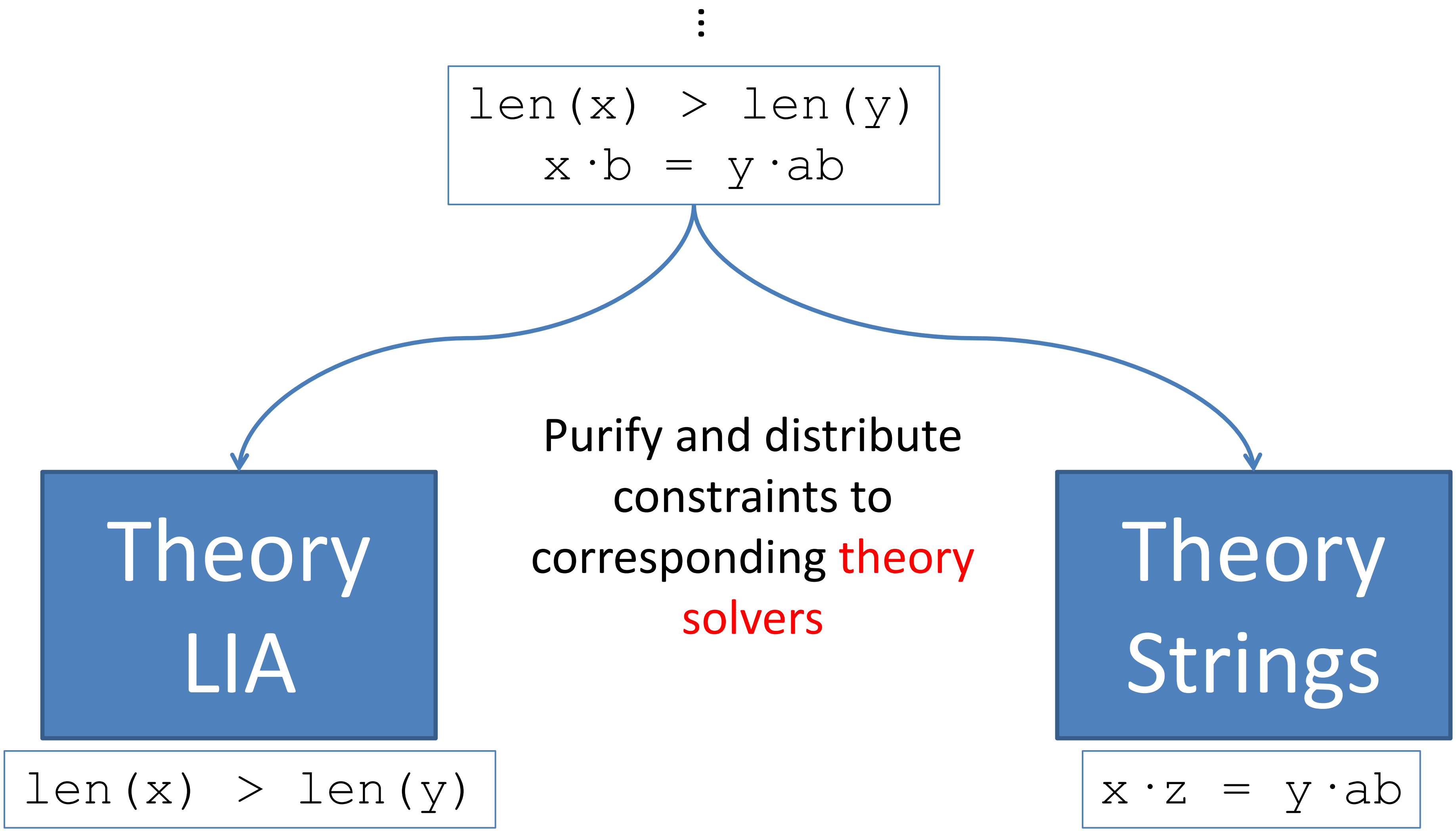
SAT Solver

Find satisfying assignment

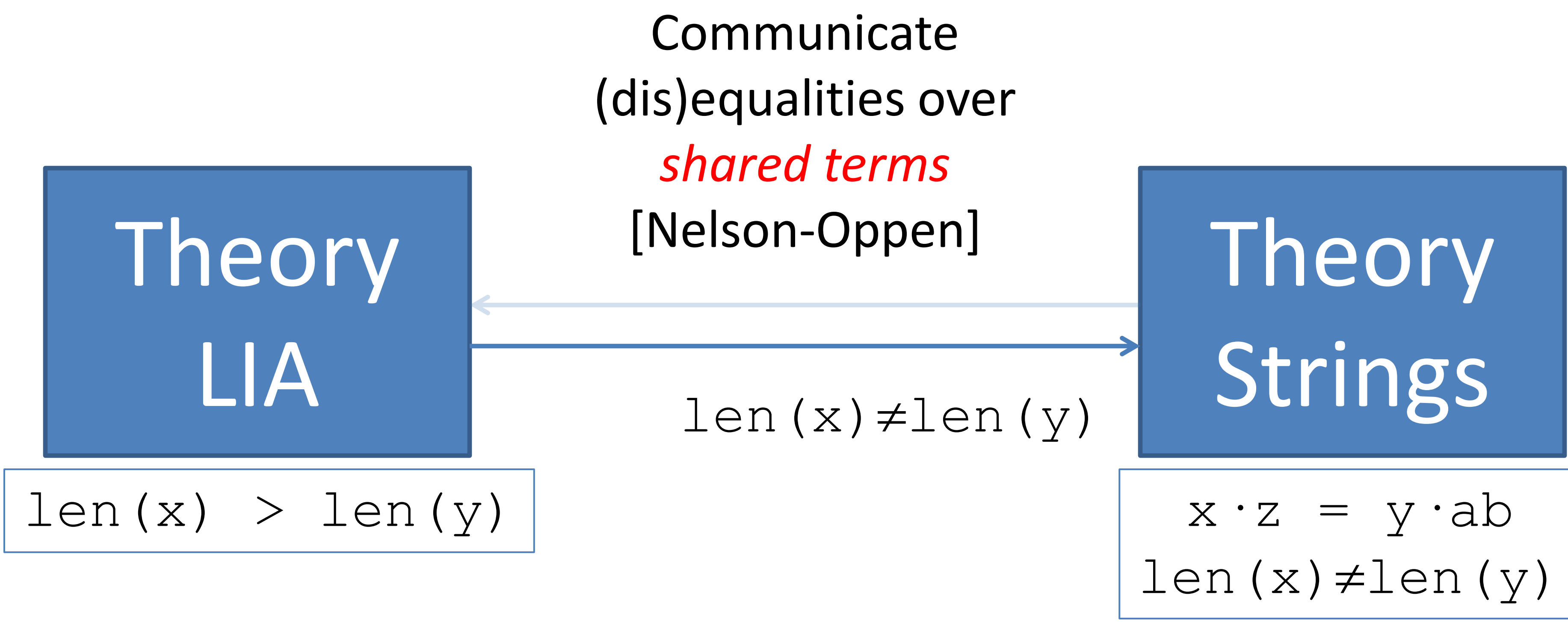
```
len(x) > len(y)
x·b = y·ab
```

⋮

DPLL(T): Cooperating *Theory Solvers*



DPLL(T): Cooperating *Theory Solvers*



DPLL(T): Cooperating *Theory Solvers*

Theory LIA

A `len(x) > len(y)`

Theory Strings

S `x·z = y·ab`

L `len(x)≠len(y)`

Summary of Approach

- Approach for $\mathtt{A} \cup \mathtt{S} \cup \mathtt{L}$ in four steps:

1. Check arithmetic constraints $\mathtt{A}$
2. Normalize equalities in $\mathtt{S}$
3. Normalize disequalities in $\mathtt{S}$
4. Check cardinality of Σ

Check Length Constraints

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

- Add equalities to A based on terms from S

Theory LIA

Theory Strings

A: `len(z)>len(w)`

S: `x·z = y·w·ab`

L: `len(x)=len(y)`, `len(z)≠len(w)`

Check Length Constraints

1. **Check length constraints**
2. **Normalize equalities**
3. **Normalize disequalities**
4. **Check cardinality of Σ**

Theory LIA

Theory Strings

A:
```
len(z)>len(w)
len(x)+len(z)=
  len(y)+len(w)+2
```

S:
```
x·z = y·w·ab
```

L:
```
len(x)=len(y)
len(z)≠len(w)
```

$\Rightarrow$ Check if A is satisfiable

Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

$$\mathtt{S}\ \left\{\ \boxed{\mathtt{x \cdot z = y \cdot w \cdot ab}}\right.$$

$$\mathtt{L}\ \left\{\ \boxed{\begin{array}{l}\mathtt{len(x) = len(y)} \\ \mathtt{len(z) \neq len(w)}\end{array}}\right.$$

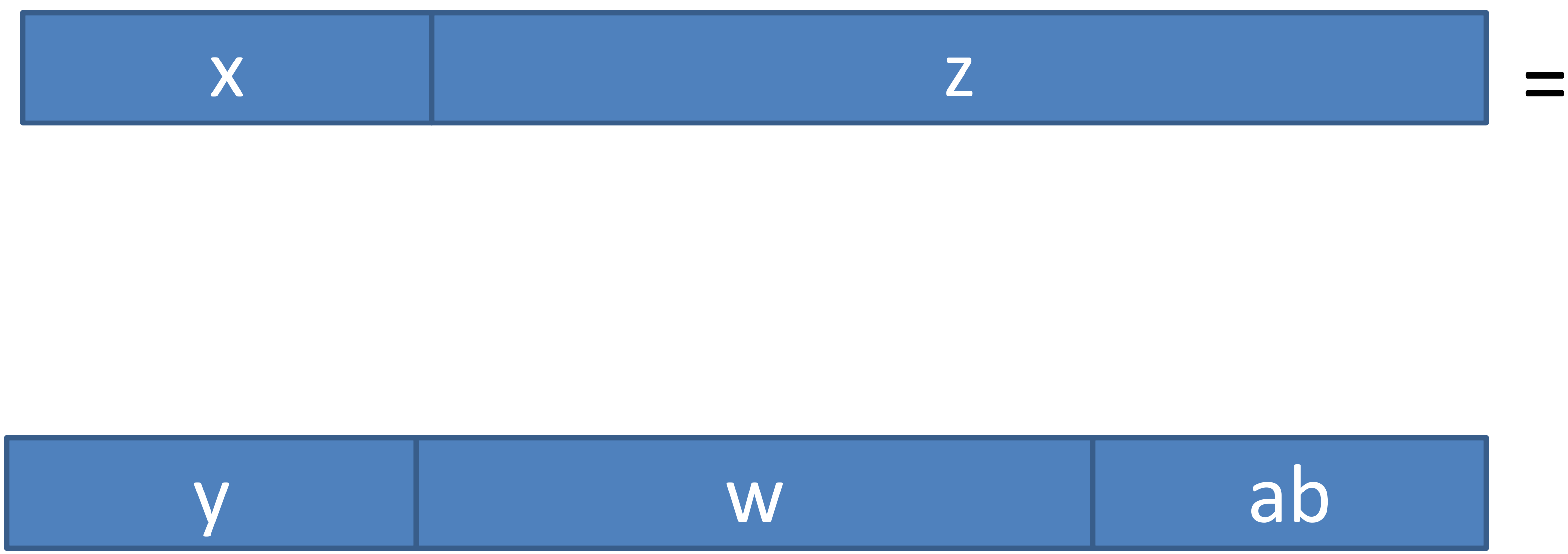
- To check satisfiability of equalities in S,
  - Add additional equalities to S
  - Until pairs of equiv. terms have same normal form

Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

S $\left\{ \boxed{\texttt{x}\cdot\texttt{z}=\texttt{y}\cdot\texttt{w}\cdot\texttt{ab}} \right.$

L $\left\{ \boxed{\begin{array}{l}\texttt{len(x)=len(y)}\\ \texttt{len(z)}\neq\texttt{len(w)}\end{array}} \right.$



Normalize Equalities

1. **Check length constraints**
2. **Normalize equalities**
3. **Normalize disequalities**
4. **Check cardinality of Σ**

S: $x \cdot z = y \cdot w \cdot ab$, $x = y$

L: $\mathrm{len}(x) = \mathrm{len}(y)$, $\mathrm{len}(z) \neq \mathrm{len}(w)$

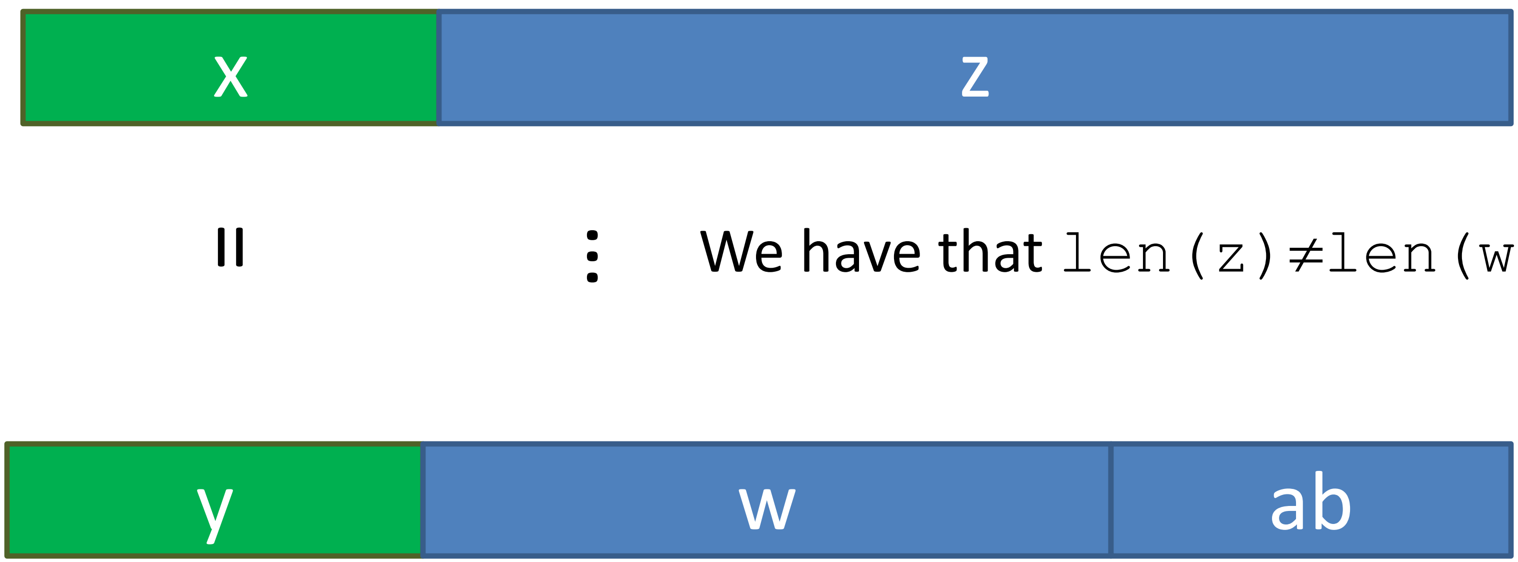
= Propagate, since $\mathrm{len}(x) = \mathrm{len}(y)$

Normalize Equalities

1. **Check length constraints**
2. **Normalize equalities**
3. **Normalize disequalities**
4. **Check cardinality of Σ**

$$S \left\{ \begin{array}{c} x \cdot z = y \cdot w \cdot ab \\ x = y \end{array} \right.$$

$$L \left\{ \begin{array}{l} \mathrm{len}(x) = \mathrm{len}(y) \\ \mathrm{len}(z) \neq \mathrm{len}(w) \end{array} \right.$$



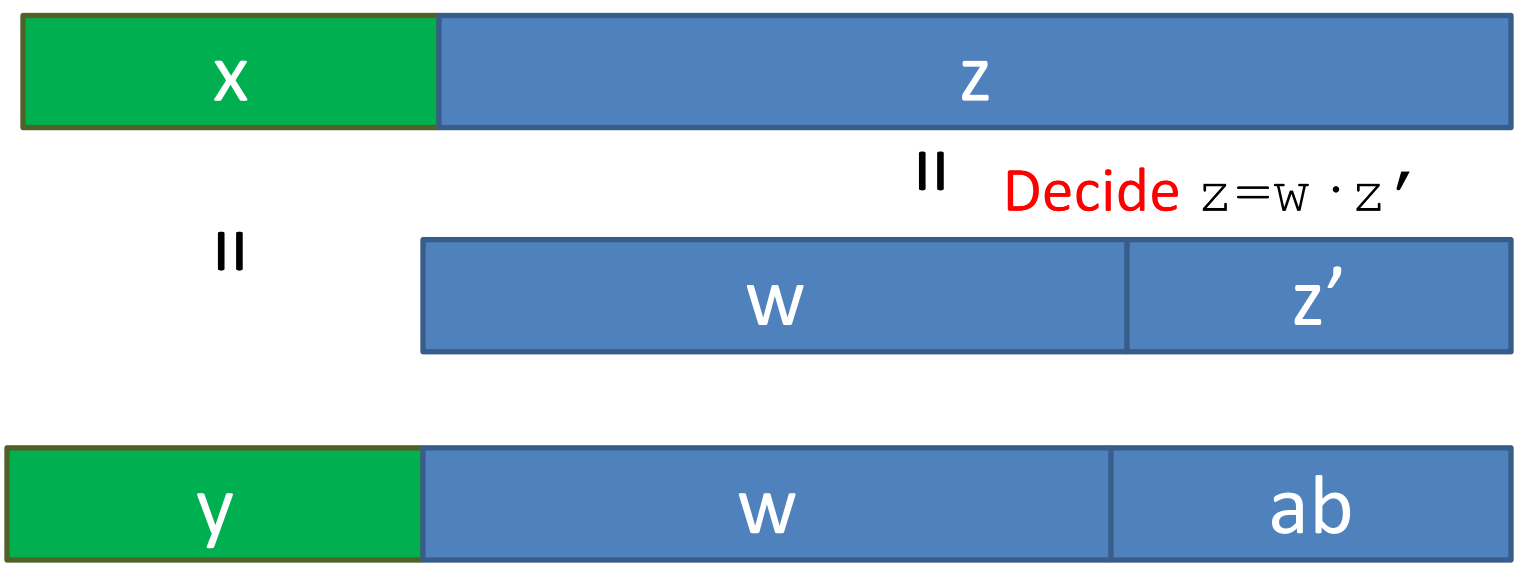
$\therefore$ We have that $\mathrm{len}(z) \neq \mathrm{len}(w)$

Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

$$S \left\{ \begin{array}{c} x \cdot z = y \cdot w \cdot ab \\ x = y \\ z = w \cdot z' \end{array} \right.$$

$$L \left\{ \begin{array}{c} \mathrm{len}(x) = \mathrm{len}(y) \\ \mathrm{len}(z) \neq \mathrm{len}(w) \end{array} \right.$$



Normalize Equalities

1. **Check length constraints**
2. **Normalize equalities**
3. **Normalize disequalities**
4. **Check cardinality of Σ**

$$S \left\{ \begin{array}{c} x \cdot z = y \cdot w \cdot ab \\ x = y \\ z = w \cdot z' \end{array} \right.$$

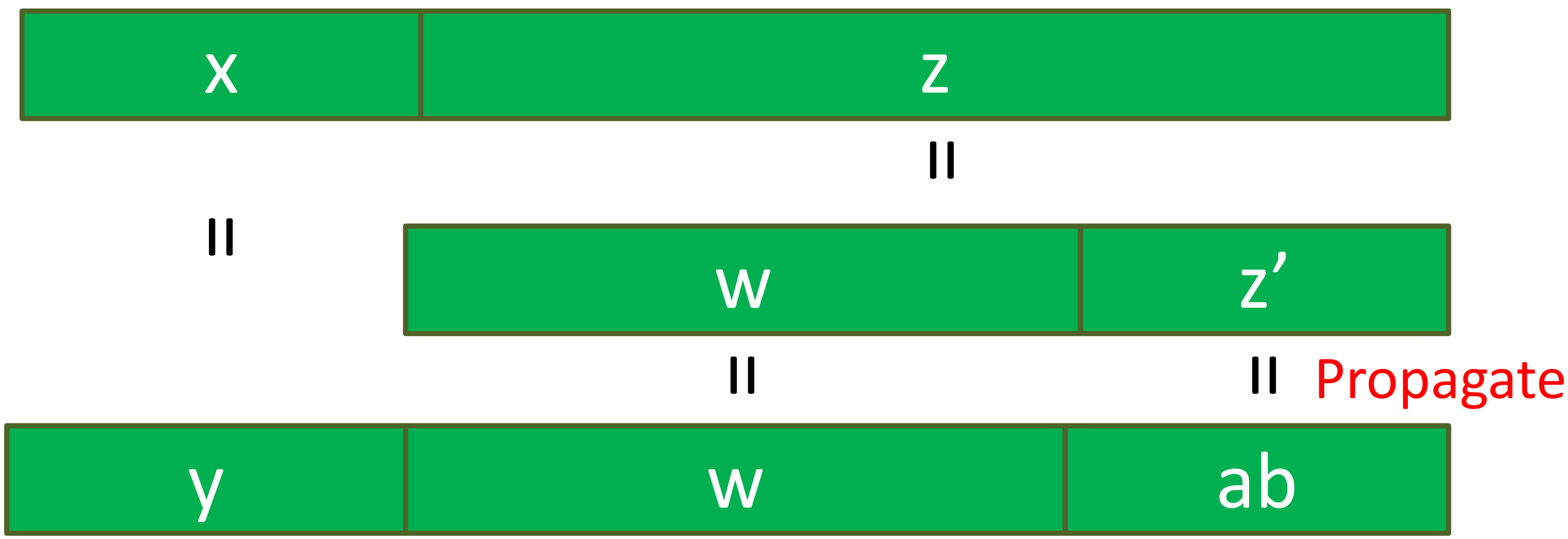
$$L \left\{ \begin{array}{l} \mathrm{len}(x) = \mathrm{len}(y) \\ \mathrm{len}(z) \neq \mathrm{len}(w) \end{array} \right.$$

Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

$$S \left\{ \begin{array}{c} \texttt{x}\cdot\texttt{z}=\texttt{y}\cdot\texttt{w}\cdot\texttt{ab} \\ \texttt{x}=\texttt{y} \\ \texttt{z}=\texttt{w}\cdot\texttt{z}' \\ \texttt{z}'=\texttt{ab} \end{array} \right.$$

$$L \left\{ \begin{array}{l} \texttt{len(x)=len(y)} \\ \texttt{len(z)}\neq\texttt{len(w)} \end{array} \right.$$



$\Rightarrow$ *Normal form*

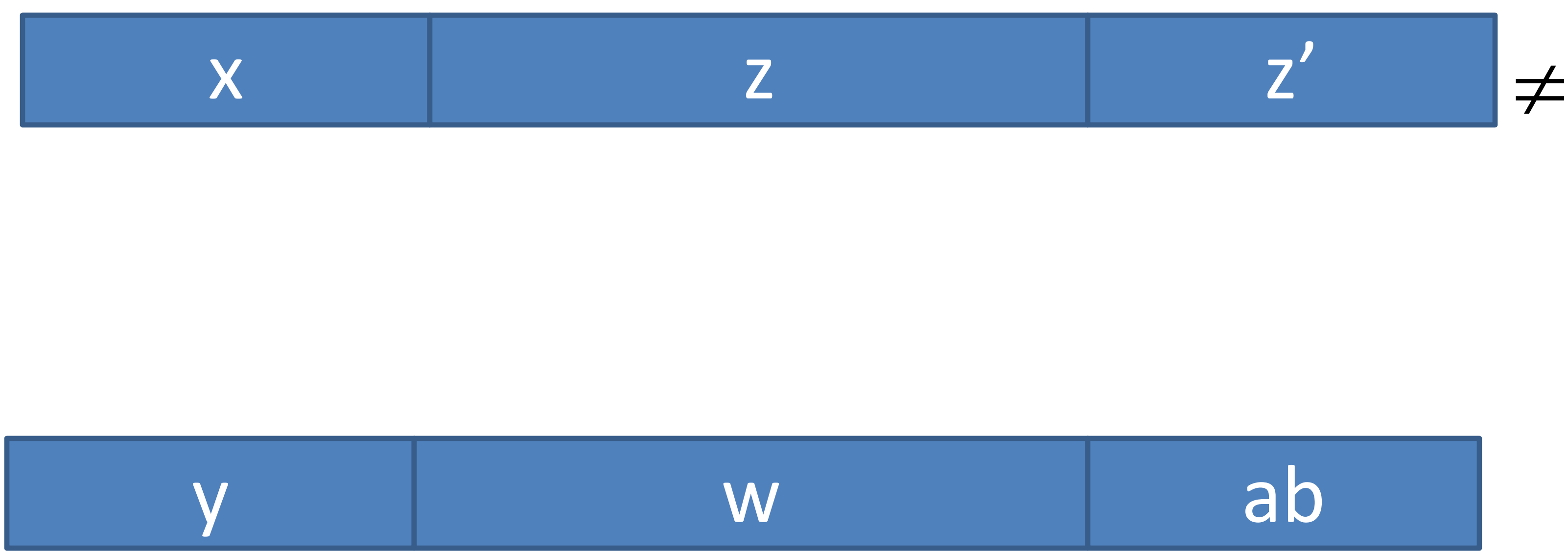
Normalize Disequalities

1. Check length constraints
2. Normalize equalities
3. **Normalize disequalities**
4. Check cardinality of Σ

- Disequalities normalized analogously

S: $x \cdot z \cdot z' \neq y \cdot w \cdot ab$, $x = y$

L: $len(x) = len(y)$, $len(z) = len(w)$

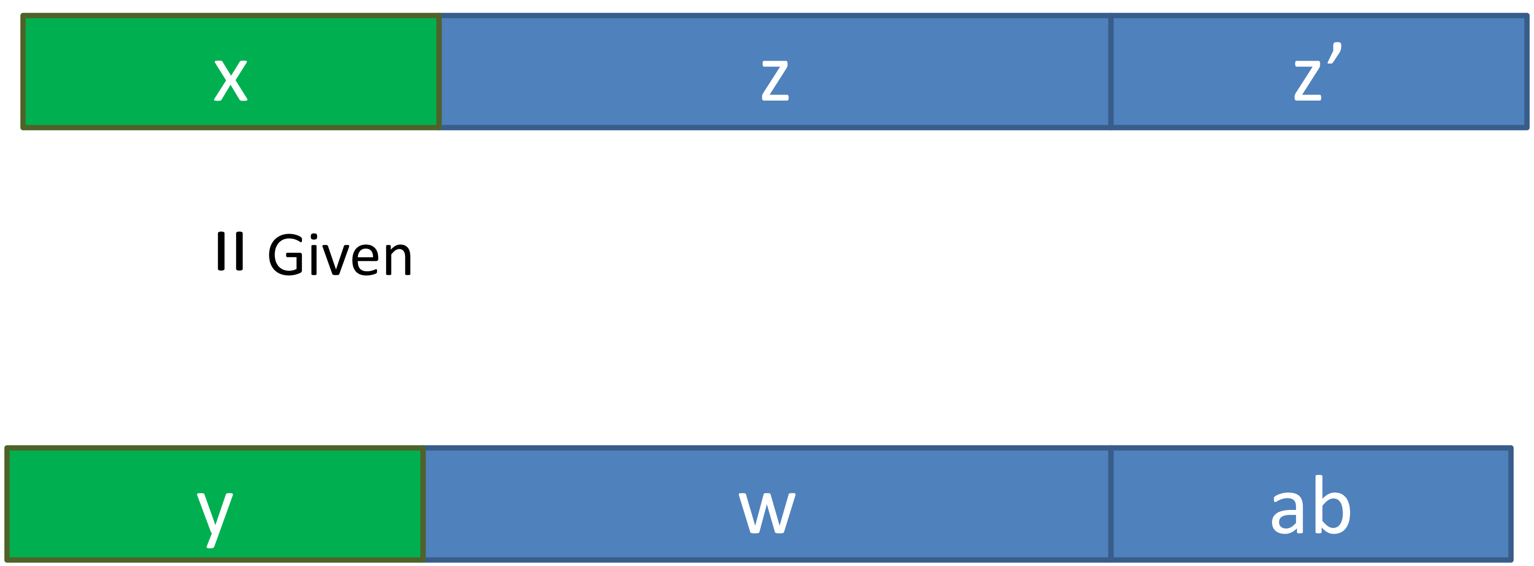


Normalize Disequalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

S: $x \cdot z \cdot z' \neq y \cdot w \cdot ab$, $x = y$

L: $len(x) = len(y)$, $len(z) = len(w)$

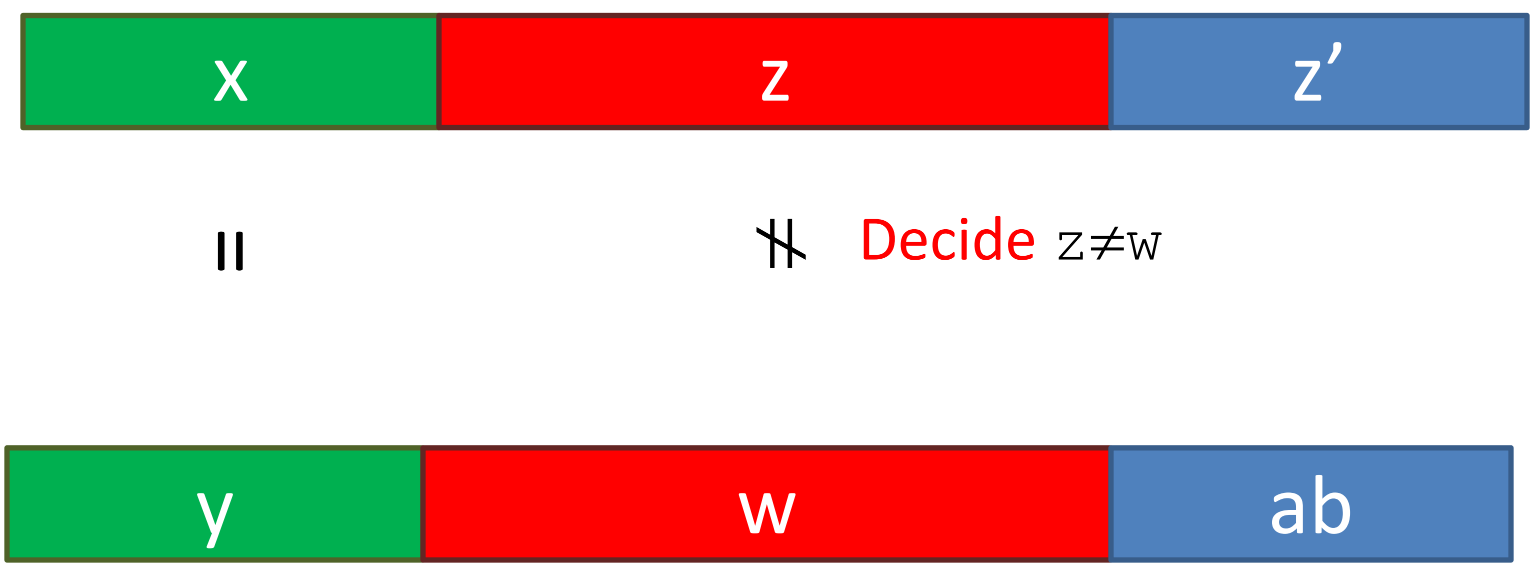


Normalize Disequalities

1. Check length constraints
2. Normalize equalities
3. **Normalize disequalities**
4. Check cardinality of Σ

$$S \left\{ \begin{array}{c} x \cdot z \cdot z' \neq y \cdot w \cdot ab \\ x = y \\ z \neq w \end{array} \right.$$

$$L \left\{ \begin{array}{l} \mathrm{len}(x) = \mathrm{len}(y) \\ \mathrm{len}(z) = \mathrm{len}(w) \end{array} \right.$$



Decide $z \neq w$

$\Rightarrow$ *Normal form*

Check Cardinality of Σ

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of Σ

- $\mathtt{S}$ may be unsatisfiable since Σ is finite
- For instance, if:
  - Σ is a finite alphabet of 256 characters, and
  - $\mathtt{S}$ entails that 257 distinct strings of length 1 exist

  Then:
  - $\mathtt{S}$ is unsatisfiable
- Performed as a last step of our procedure

Rule-Based Procedure

...

$$\text{F-Unify}\ \frac{\mathsf{F}\,s = (w, u, u_1) \quad \mathsf{F}\,t = (w, v, v_1) \quad s \approx t \in \mathcal{C}(\mathsf{S}) \quad \mathsf{S} \models \mathsf{len}\,u \approx \mathsf{len}\,v}{\mathsf{S} := \mathsf{S}, u \approx v}$$

$$\text{F-Split}\ \frac{\begin{array}{c}\mathsf{F}\,s = (w, u, u_1) \quad \mathsf{F}\,t = (w, v, v_1) \quad s \approx t \in \mathcal{C}(\mathsf{S}) \quad \mathsf{S} \models \mathsf{len}\,u \not\approx \mathsf{len}\,v \\ u \notin \mathcal{V}(v_1) \quad v \notin \mathcal{V}(u_1)\end{array}}{\mathsf{S} := \mathsf{S}, u \approx \mathsf{con}(v, z) \quad \| \quad \mathsf{S} := \mathsf{S}, v \approx \mathsf{con}(u, z)}$$

$$\text{F-Loop}\ \frac{\mathsf{F}\,s = (w, x, u_1) \quad \mathsf{F}\,t = (w, v, v_1, x, v_2) \quad s \approx t \in \mathcal{C}(\mathsf{S}) \quad x \notin \mathcal{V}((v, v_1))}{\begin{array}{c}\mathsf{S} := \mathsf{S},\ x \approx \mathsf{con}(z_2, z),\ \mathsf{con}(v, v_1) \approx \mathsf{con}(z_2, z_1),\ \mathsf{con}(u_1) \approx \mathsf{con}(z_1, z_2, v_2) \\ \mathsf{R} := \mathsf{R}, z \ \mathsf{in}\ \mathsf{star}(\mathsf{set}\ \mathsf{con}(z_1, z_2)) \quad \mathsf{C} := \mathsf{C}, t\end{array}}$$

...

- Approach is algebraic
  - Rules model interaction of string + arithmetic solvers
    - A closed derivation tree ⇒ problem is UNSAT
    - A state where no rule applies ⇒ problem is SAT

Theoretical Results

- Our approach is:
  - Refutation sound
    - When it answers “UNSAT”, it can be trusted
      - Even for strings of unbounded length
  - Solution sound
    - When it answers “SAT”, it can be trusted
- (A version of) our approach is:
  - Solution complete
    - When problem is “SAT”, it will eventually find a model
      - Somewhat trivially, by finite model finding
- Our approach is:
  - Refutation incomplete
    - When problem is “UNSAT”, it is not guaranteed to derive refutation

Experimental Results

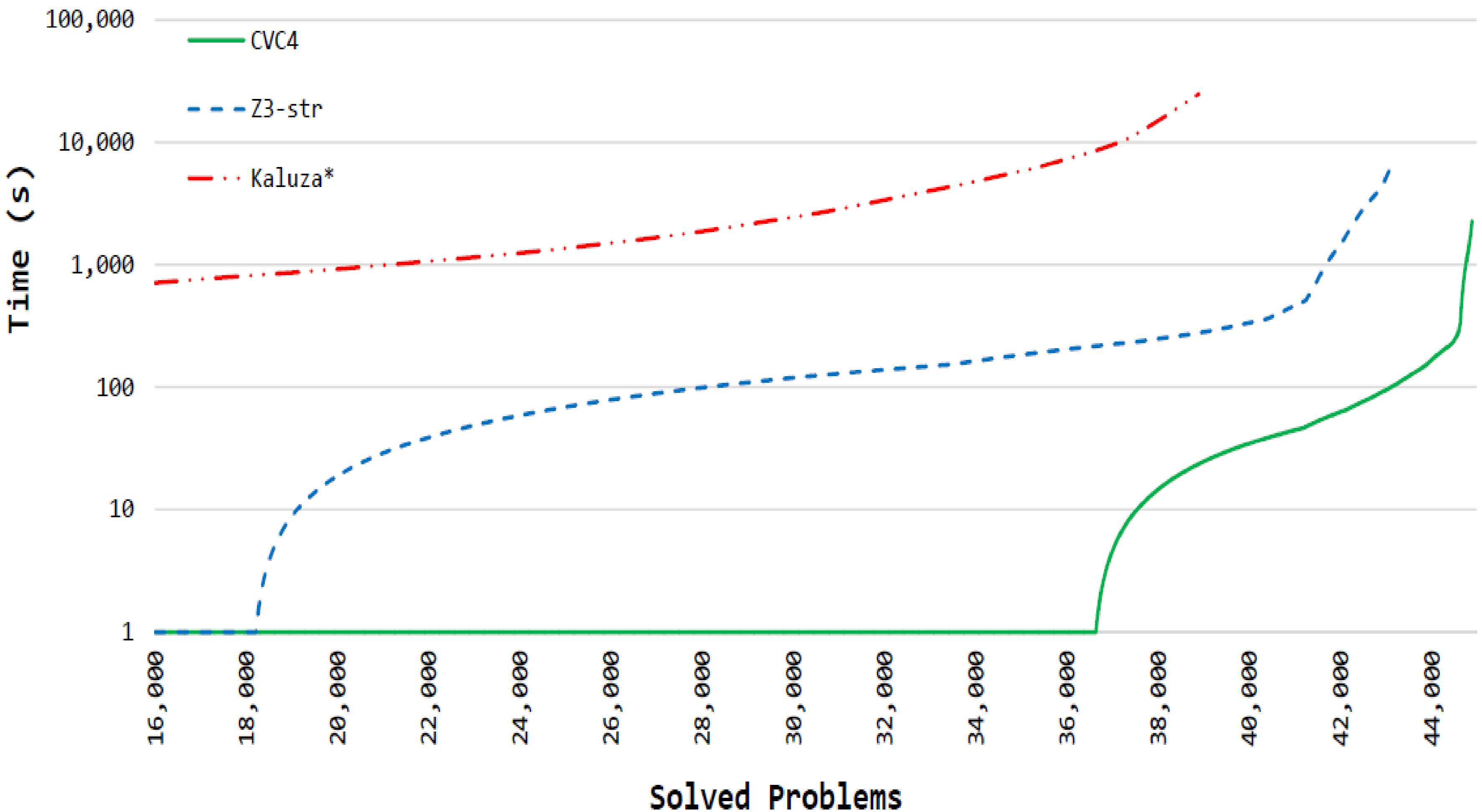
- Implemented in SMT solver CVC4
- Tested:
  - 50,000 benchmarks from Kudzu
    - Correspond to VCs in web security applications
- Compared against solvers:
  - Kaluza (UBerkeley)
  - Z3-STR (Purdue, Waterloo)

Experimental Results

	CVC4	Z3-STR		Kaluza	
Result		Incorrect[3]		Incorrect[3]	
unsat	11,625[1]	317	11,769[2]	7,154	13,435[2]
sat	33,271	1,583	31,372	n/a[4]	25,468[4]
unknown	0		0		3
timeout	2,388		2,123		84
error	0		120[5]		1,140

1. For the problems where CVC4 answers UNSAT, neither Z3-STR nor Kaluza answer SAT
2. We cannot verify the problems where CVC4 does not answer UNSAT
3. We verified these errors by asserting a model back as assertions to the tool
4. We cannot verify these answers due to bugs in Kaluza's model generation
5. One is because of non-trivial regular expression, and 119 are because of escaped characters

Experimental Results



Further Work

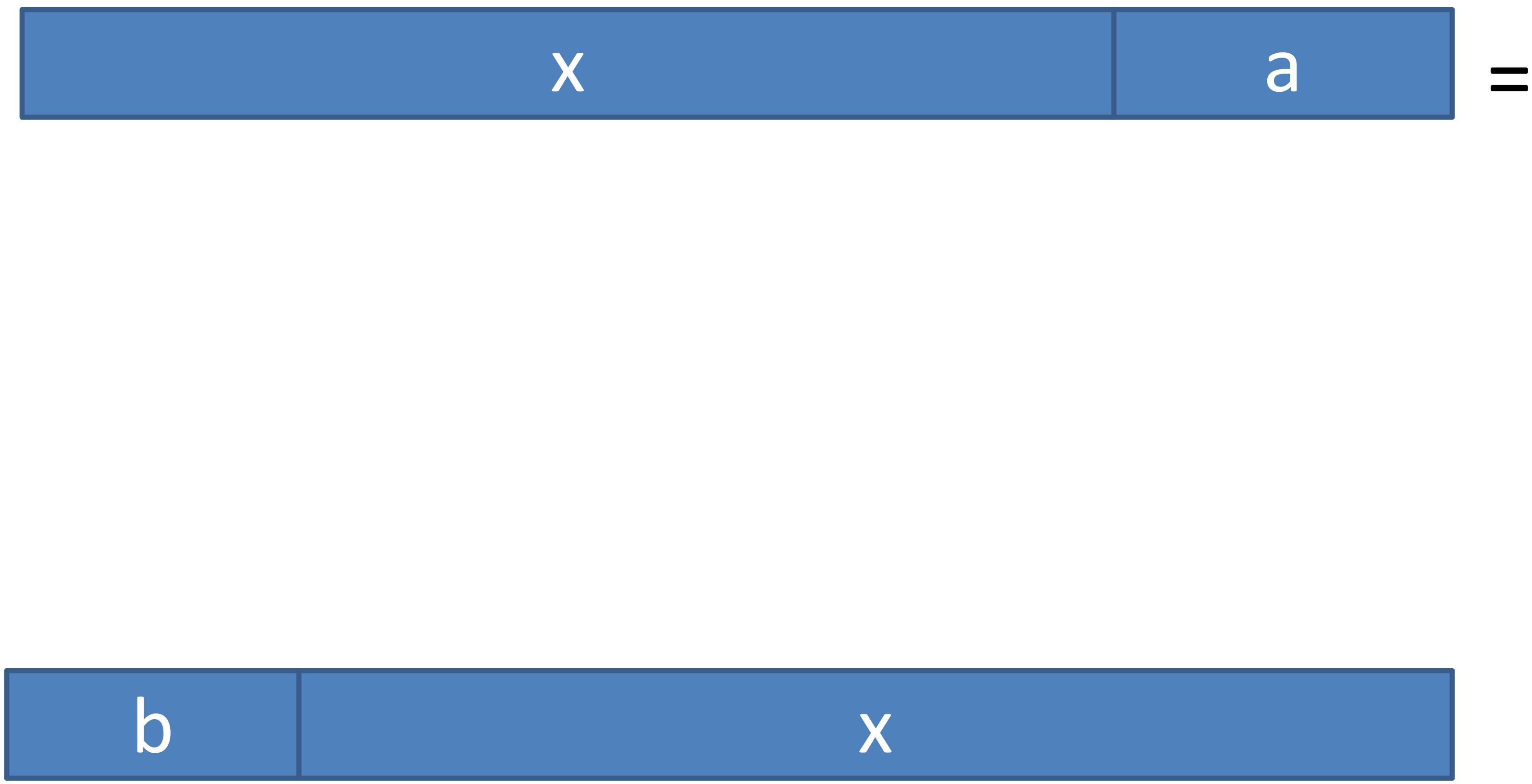
- Theoretical:
  - Identify fragments when approach is refutation complete
    - [Abdullah et al CAV14]
- Regular language membership $t \in R^*$
  - Currently handled, but naively (unrolling)
- More functions
  - `substr`, `contains`, `replace`, `prefixOf`, `suffixOf`, `str.indexOf`, `str.to.int`, `int.to.str`
- Generalize to theory of sequences

Thank You!

- CVC4 is publicly available at:
  `http://cvc4.cs.nyu.edu/`

Challenge: Looping Word Equations

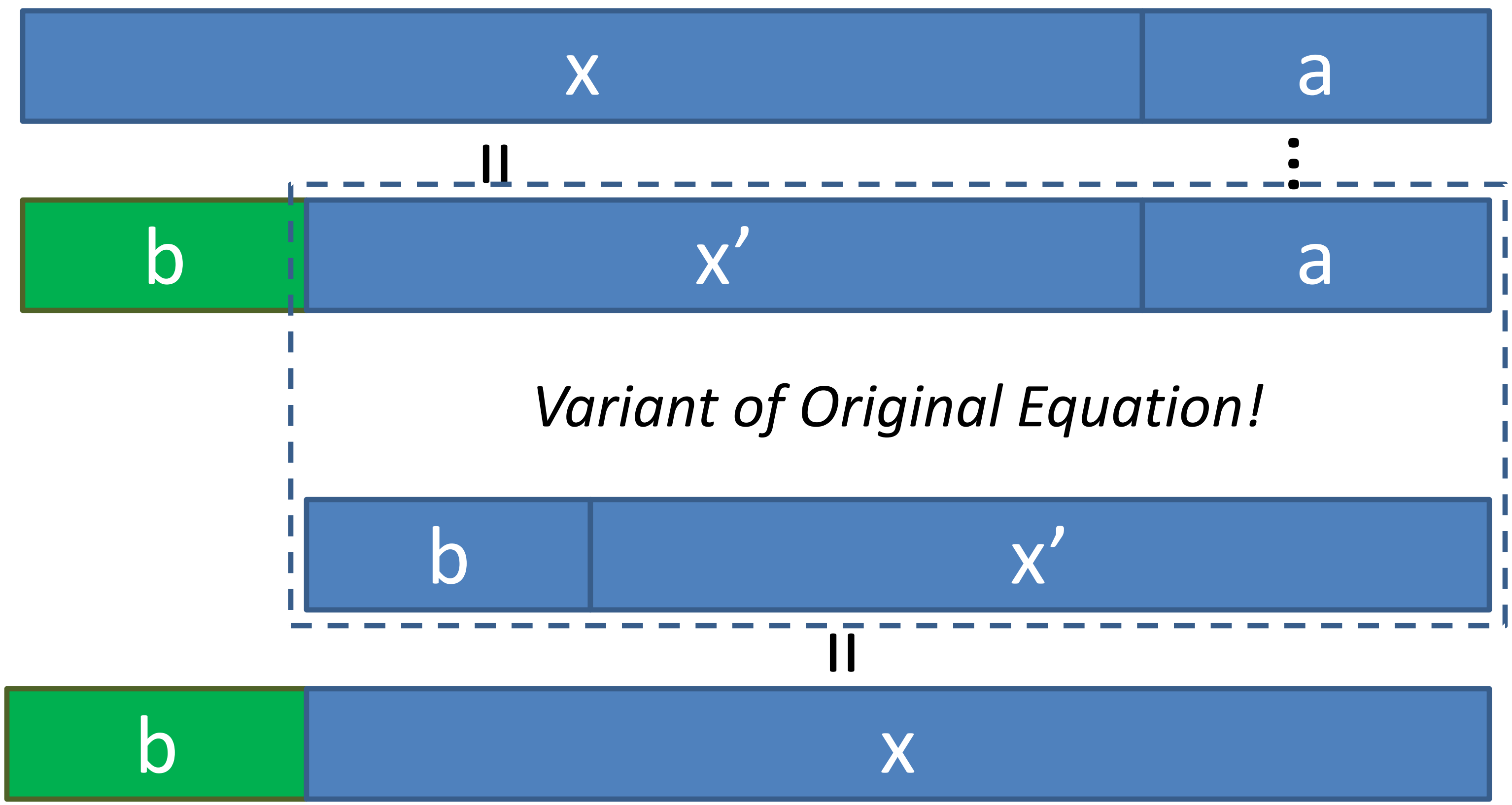
- Say we are given: $\boxed{x \cdot a = b \cdot x}$



Challenge: Looping Word Equations

$$x \cdot a = b \cdot x$$

$$x = b \cdot x'$$



Challenge: Looping Word Equations

$$\boxed{x \cdot a = b \cdot x}$$

- Solution:

  - Recognize when these cases occur

  - Reduce to regular language membership:

$$x \cdot a = b \cdot x \Leftrightarrow \exists yz.(a = y \cdot z \ \wedge \ b = z \cdot y \ \wedge \ x \in (z \cdot y)^* z)$$