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

Tianyi Liang
Andrew Reynolds
Cesare Tinelli
Morgan Deters
Clark Barrett

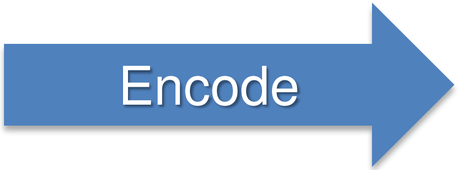
Motivation : Security Applications

```
char buff[15];
char pass;

std::cout << "Enter the password :";
gets(buff);

if (std::regex_match(
        buff,
        std::regex("([A-Z]+)") )) {
   if(strcmp(buff, “PASSWORD")) {
      std::cout << "Wrong Password";
   }
   else {
      std::cout << "Correct Password";
      pass = 'Y';
   }
}

if(pass == 'Y') {
   /* Grant the root permission*/
}
```



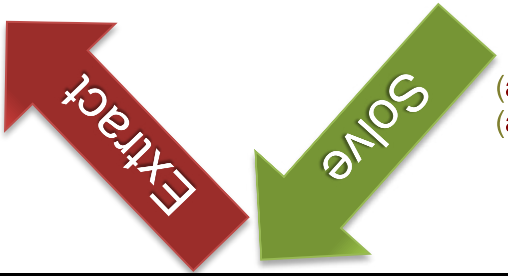

```
(set-logic QF_S)

(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 (= 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"))
```




```
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, …
- Theoretical complexity of:
  - Word equation problem is in PSPACE
  - …with length constraints is OPEN
  - …with extended functions (e.g. `replace`) is UNDECIDABLE
- Instead, focus on:
  - Solver that is efficient in practice
  - Tightly integrated into SMT solver architecture
    - Conflict analysis, T-propagation, lemma learning, …

Core Language for Theory of Strings

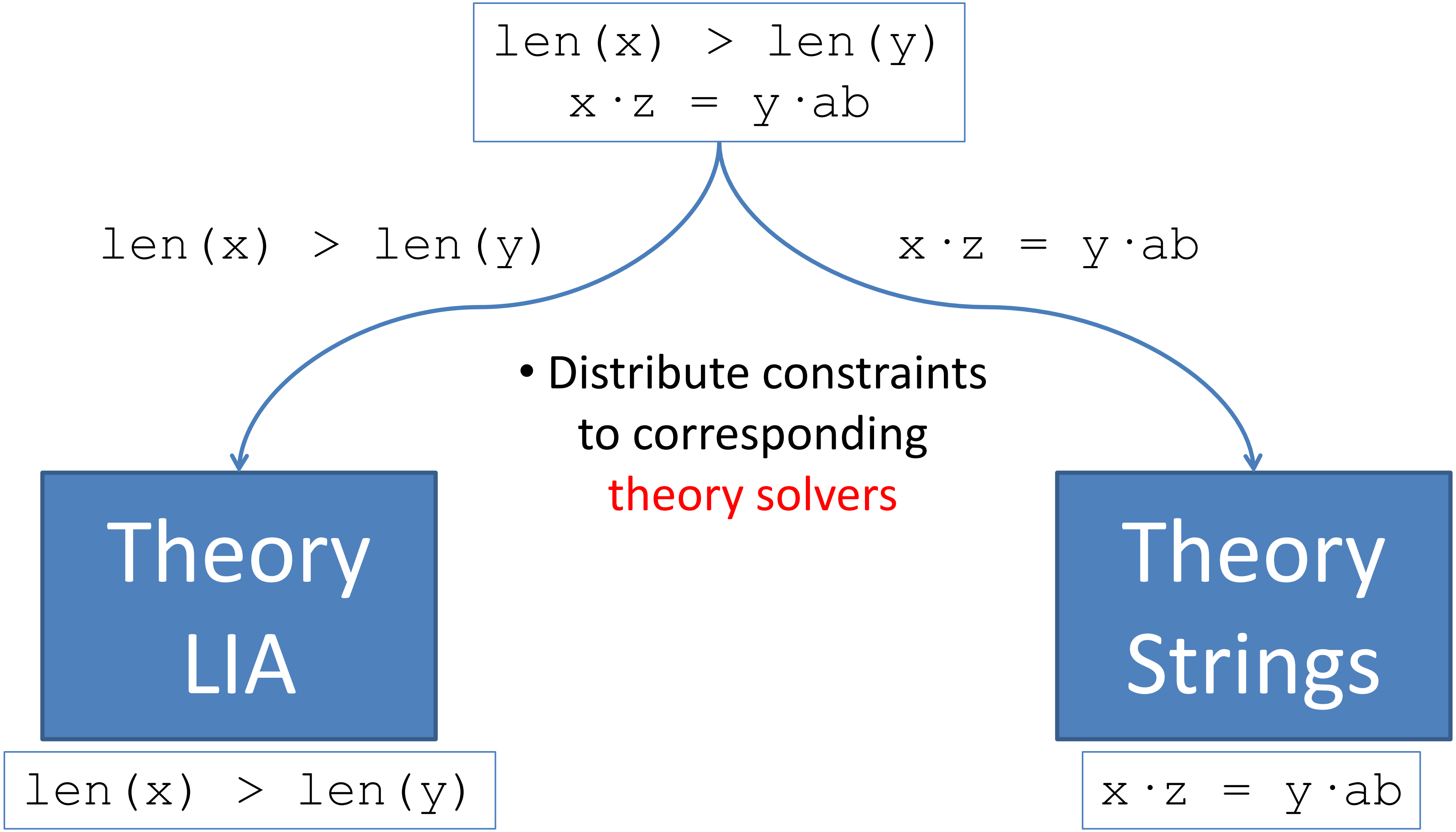
- Terms are:
  - Constants from a fixed finite alphabet Σ^* (a, ab, cbc...)
  - Free constants or "variables" (x, y, z...)
  - String concatenation

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

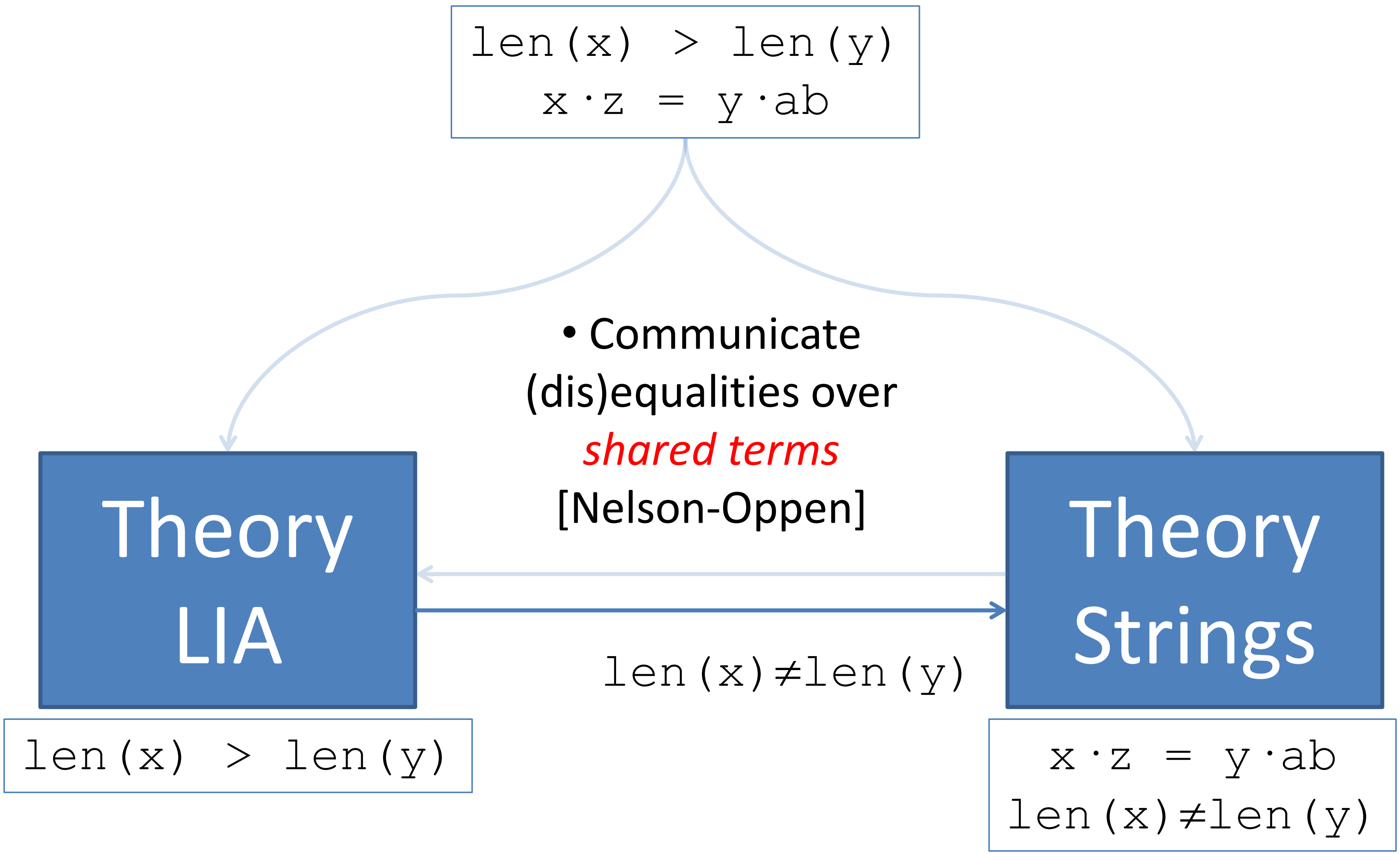
    `len(_)` : String $\rightarrow$ Int
- Example input:

```
len(x) > len(y)
   x·z = y·ab
```

Cooperating *Theory Solvers*



Cooperating *Theory Solvers*



Summary of Approach

- Determines satisfiability of $\mathtt{A} \cup \mathtt{S}$, where
  - $\mathtt{A}$ is a set of linear arithmetic constraints
  - $\mathtt{S}$ is a set of (dis)equalities over:
    - String terms
    - Length terms

$$\mathtt{x \cdot z = y \cdot ab}$$
$$\mathtt{len(x) \neq len(y)}$$

- Uses procedure consisting of four steps:

1. Check length 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 regarding the length of (non-variable) terms from S

Theory LIA

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

Theory Strings

S: `len(x) ≠ len(y)`, `x·z = y·ab`

Check Length Constraints

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

Theory LIA

Theory Strings

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

S: len(x) $\neq$ len(y)
x·z = y·ab

$\Rightarrow$ Check if A is satisfiable

Normalize Equalities

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

- To show: satisfiability of (dis)equalities S between string terms

Theory Strings

len(x) ≠ len(y)
x·z = y·ab

- To ensure equality t=s has model:
  - If t and s are non-variable,
    - Must be equivalent to flat forms F[t], F[s]
      - F[t] and F[s] are syntactically equivalent
- Flat form F[t] computed by expanding/flattening t

Normalize Equalities

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

- Modified example:

```
len(x) = len(y)
z·w = y·ab
z = x·a
```

- Flat form of terms from first equality are not the same:
  - F[z·w] is: x·a·w
  - F[y·ab] is: y·ab
- Procedure continues based on three cases:
  - We know the length of x and y are equal : *conclude x=y*
  - We know the length of x and y are disequal : conclude
    ∃k.((x=y·k ∨ y=x·k) ∧ len(k)>0)
  - We know neither : guess their lengths are equal, restart

Normalize Equalities

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

- After concluding x=y,

$$\mathrm{len}(x) = \mathrm{len}(y)$$
$$z \cdot w = y \cdot ab$$
$$z = x \cdot a$$
$$x = y$$

- Flat form of terms from first equality are now, e.g.:
  - F[z·w] is: y·a·w
  - F[y·ab] is: y·ab
- Will conclude w=b, after which F[z·w]=F[y·ab]

Normalize Equalities

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

- For $t=s$, procedure makes progress* towards:
  - Towards forcing flat forms $F[t]$ and $F[s]$ equal, or
  - Discovering conflicts
- If $F[t_1]=\ldots=F[t_n]$ for an eq class $E=\{t_1 \ldots t_n\}$:
  - We refer to $F[t_1]$ as the normal form $N[t_1]$ of E
- If normal form exists for each eq class,
  - Then a model exists for all equalities from S
    - Constructed trivially, given normal form

* exception: looping word equations (explained later)

Normalize Disequalities

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

- For disequalities in S
  - A disequality $t \neq s$ is normalized if:
    - $len(t) \neq len(s)$, or
    - $N[t] = t_1 \cdot u \cdot t_2$ and $N[s] = s_1 \cdot v \cdot s_2$, where:
      - $len(t_1) = len(t_2)$,
      - $len(u) = len(v)$, and
      - $u \neq v$
- For example:

$$len(z) \neq len(y)$$ ✓

$$z \neq y$$ ✓

$$x \cdot a \cdot z \neq x \cdot b \cdot z$$ ✓

$$x \cdot w \neq y \cdot b$$ ✗

Normalize Disequalities

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

- To normalize disequalities,
  - Proceed by cases, similar to Step 2
    - In example, we would succeed, for example if:
      - $\text{len}(x \cdot w) \neq \text{len}(y \cdot b)$, or
      - $\text{len}(x) = \text{len}(y)$ and $x \neq y$,
      - …
  - Continue until all disequalities are normalized

$$\text{len}(z) \neq \text{len}(y)$$
$$z \neq y$$
$$x \cdot a \cdot z \neq x \cdot b \cdot z$$
$$x \cdot w \neq y \cdot b$$

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

Challenge: Looping Word Equations

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

Challenge: Looping Word Equations

- Say we are given: $x \cdot a = b \cdot x$
- Flat forms are:

  $F[x \cdot a] = x \cdot a$

  $F[b \cdot x] = b \cdot x$

- Compare $len(x)$ and $len(b)$, i.e. 1
  - If $len(x)=1$, then $x=a$ and $x=b \Rightarrow$ conflict
  - If $len(x) \neq 1$
    - If x is a prefix of b (i.e. it is empty), then $a=b \Rightarrow$ conflict
    - If b is a prefix of x, then $x=b \cdot k$ for some k

Challenge: Looping Word Equations

- Now we have:

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

- Flat forms of first equation are:

$$F[x \cdot a] = b \cdot k \cdot a$$
$$F[b \cdot x] = b \cdot b \cdot k$$

$\Rightarrow$ Problem: looping!

- Solution:
  - Recognize when these cases occur
  - Reduce to regular language membership:

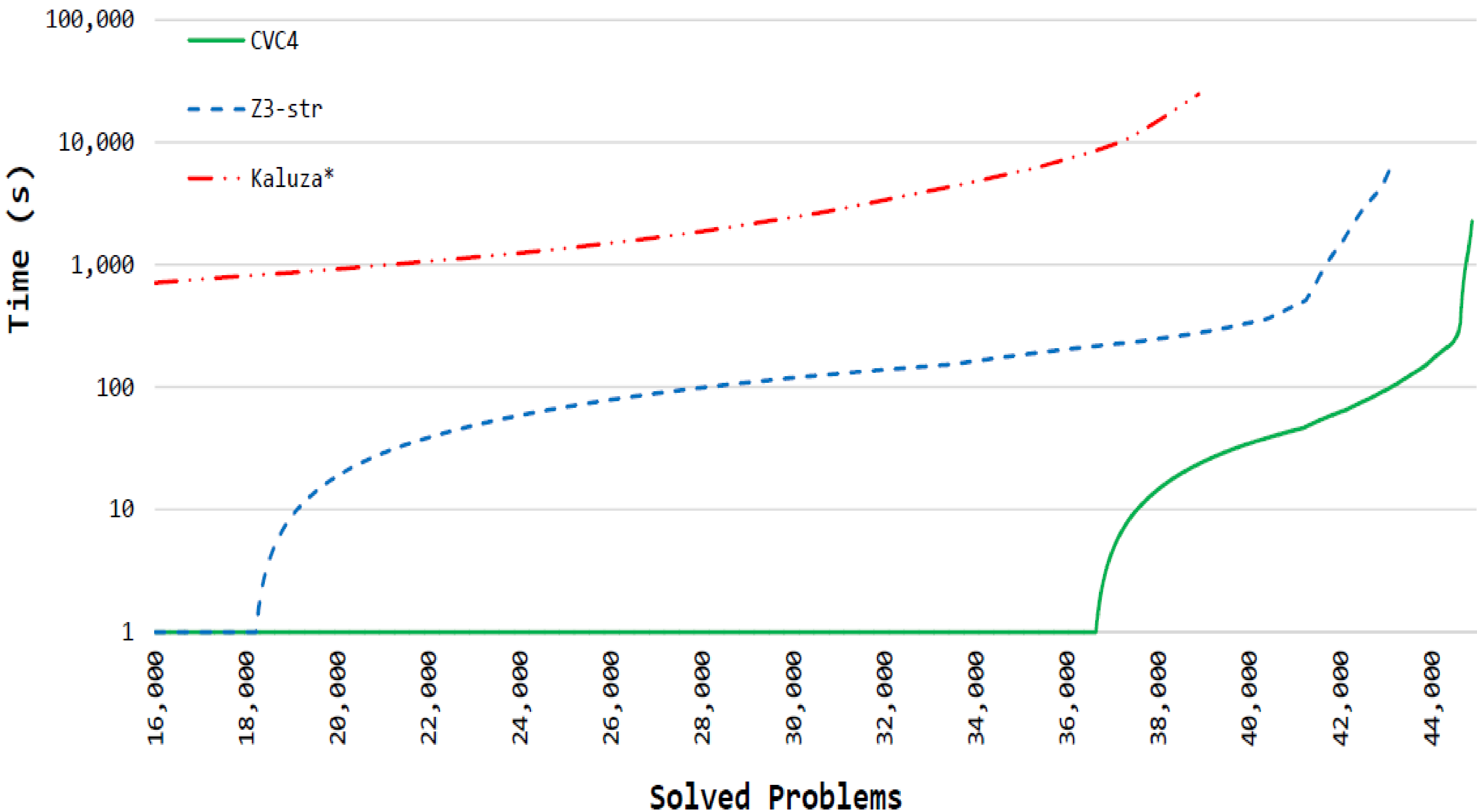
$$x \cdot a = b \cdot x \Leftrightarrow \exists yz. (a = y \cdot z \wedge b = z \cdot y \wedge x \in (z \cdot y)^* z)$$

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



Theoretical Results

- Our approach is:
  - Refutation sound
    - When it answer "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 it is "SAT", it will eventually get a model
      - Somewhat trivially, by finite model finding
- Our approach is not:
  - Refutation complete
    - When it is "UNSAT", it is not guaranteed to derive refutation
      - Would like to identify fragments (i.e. non-cyclical) where it is

Further Work

- Handling regular language membership `t∈R*`
  - Currently handled, but naively (unrolling)
- Handling extended functions
  - `substr`, `contains`, `replace`, `prefixOf`, `suffixOf`, `str.indexOf`, `str.to.int`, `int.to.str`
  - Many are challenging, for instance:

$$\neg\texttt{contains}(x,y)$$

    - Intuitively, requires (universal) quantification over the positions of x

Questions?

- For more details, see CAV 2014 paper
- CVC4 is publicly available at:

  `http://cvc4.cs.nyu.edu/web/`