Designing a Fast and Trustworthy String Solver

Andrew Reynolds & Cesare Tinelli

MOSCA 2023

THE UNIVERSITY OF IOWA

Satisfiability Modulo Theories (SMT) Solvers

Many applications:

- Software verification
- Symbolic execution
- Security analysis
- Theorem proving

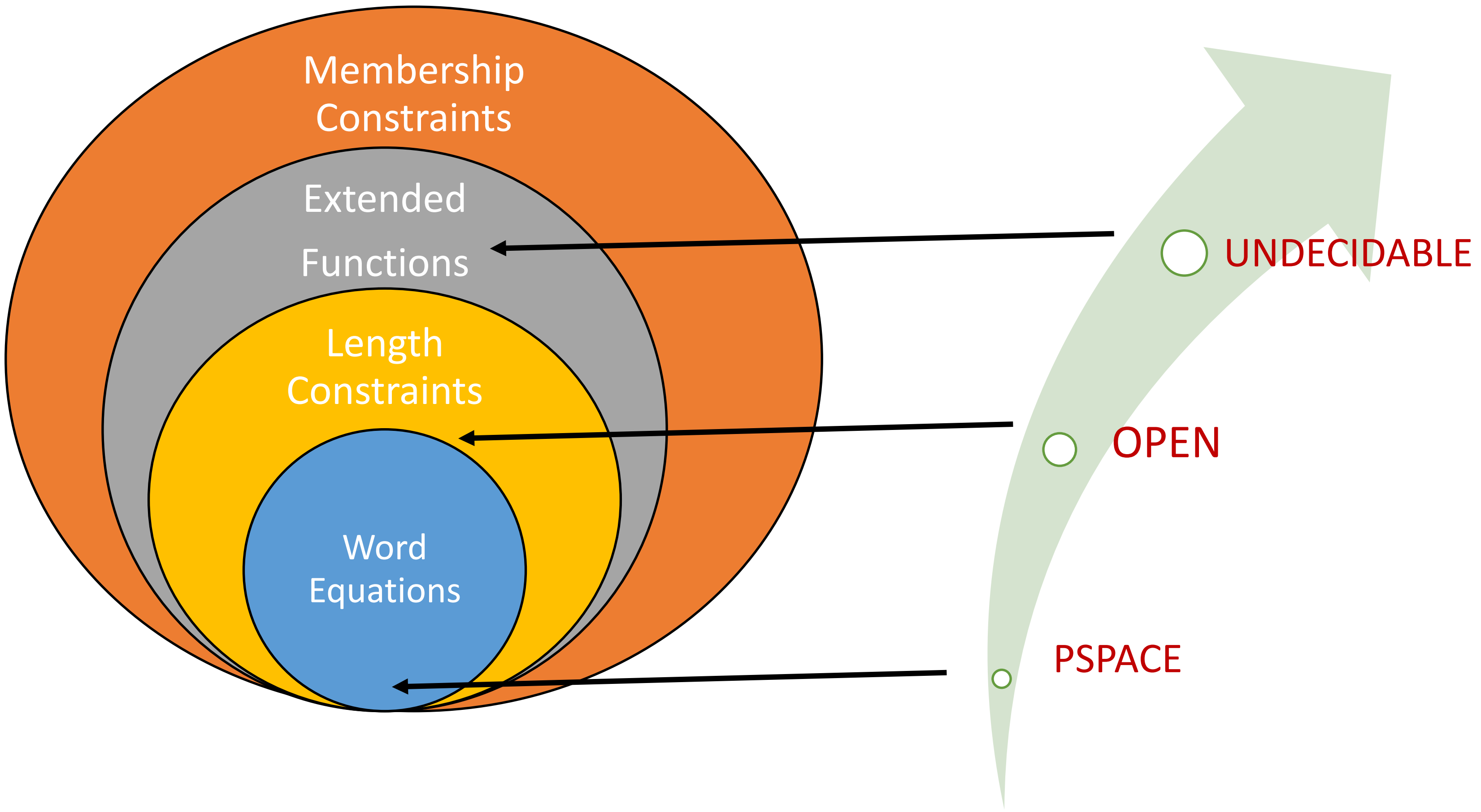
Traditionally:

- Efficient solvers for quantifier-free constraints over (combinations of) theories
  - Arithmetic, Arrays, Bit vectors

In this talk:

- SMT techniques for string and RE constraints

Strings and RE: Theoretical Challenges



Membership Constraints

Extended Functions

Length Constraints

Word Equations

UNDECIDABLE

OPEN

PSPACE

Many applications require *extended string functions* and *RE memberships*

Ex.: $\mathrm{toInt}(x) \neq 44$, $\mathrm{toLower}(x) = \mathbf{abc}$, $x \in \mathrm{range}(\mathbf{A}, \mathbf{Z})$

The CVC4 and cvc5 SMT Solvers

Support for many theories and features

- UF, (non)linear arithmetic, arrays
- Bit-vectors, floating points
- (Multi)sets, relations, datatypes
- Strings and regular expressions

Stanford
University

THE UNIVERSITY
OF IOWA

Co-developed at Stanford and Iowa

Project Leaders: Clark Barrett, Cesare Tinelli

String solver developers: Andrew Reynolds, Andres Noetzli

SMT Solvers for Strings: Timeline

	2009	2012	2014	2017	2019	2020	2022
	CVC4 project begins	CVC4 1.0 release	CVC4 support for strings + RE	Support for Extended String Functions	Aggressive Rewriting	Support For Proofs	cvc5 1.0 release
CVC papers on strings			**[CAV 14]**	**[CAV 17]**	**[CAV 19]**	**[FMCAD 20]** **[IJCAR 20]**	**[CAV 22]** **[FMCAD 22]**

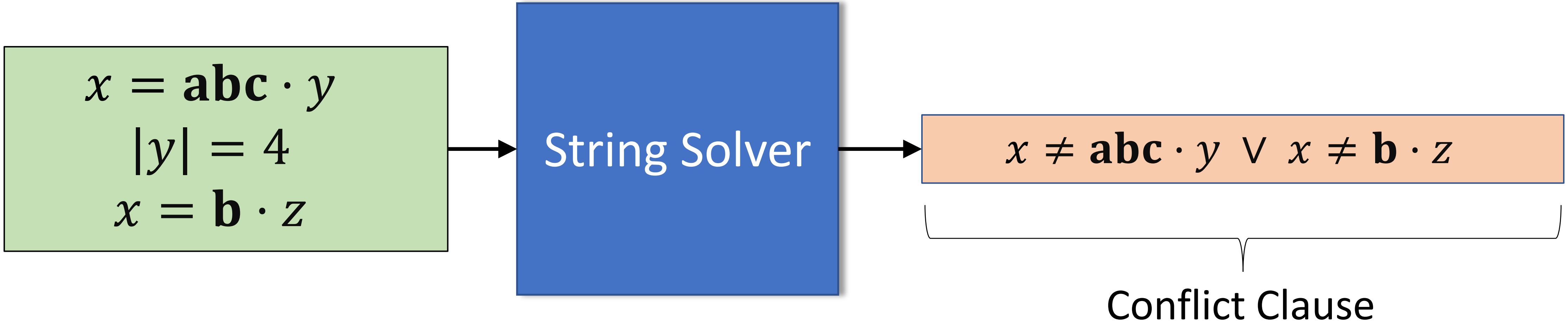
Other SMT String Solvers

Z3Str   S3   Z3Str2   Z3Seq   Trau   Z3Str3   Z3Str4   ….

Norn   OSTRICH

String Reasoning in cvc5 in a Nutshell

A Theory Solver for Strings [Liang et al., CAV’14]



$x = \mathbf{abc} \cdot y$

$|y| = 4$

$x = \mathbf{b} \cdot z$

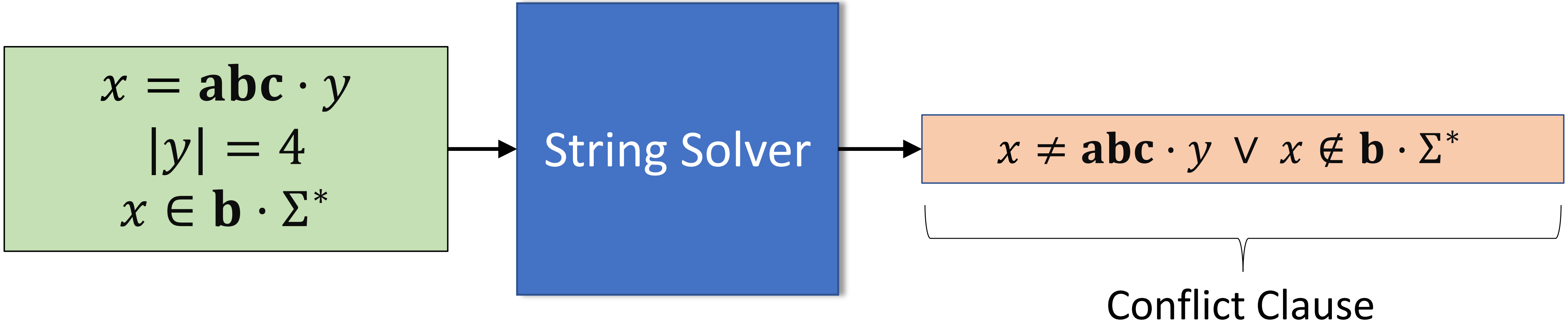
String Solver

$\underbrace{x \neq \mathbf{abc} \cdot y \lor x \neq \mathbf{b} \cdot z}_{\text{Conflict Clause}}$

Designed a string solver for concat + length + RE constraints that is:

- refutation and model sound (“unsat” and “sat” can be trusted)
- not terminating in general
- efficient in practice

Perfecting support for REs [Liang et al., FroCoS’15]



$$x = \mathbf{abc} \cdot y$$
$$|y| = 4$$
$$x \in \mathbf{b} \cdot \Sigma^*$$

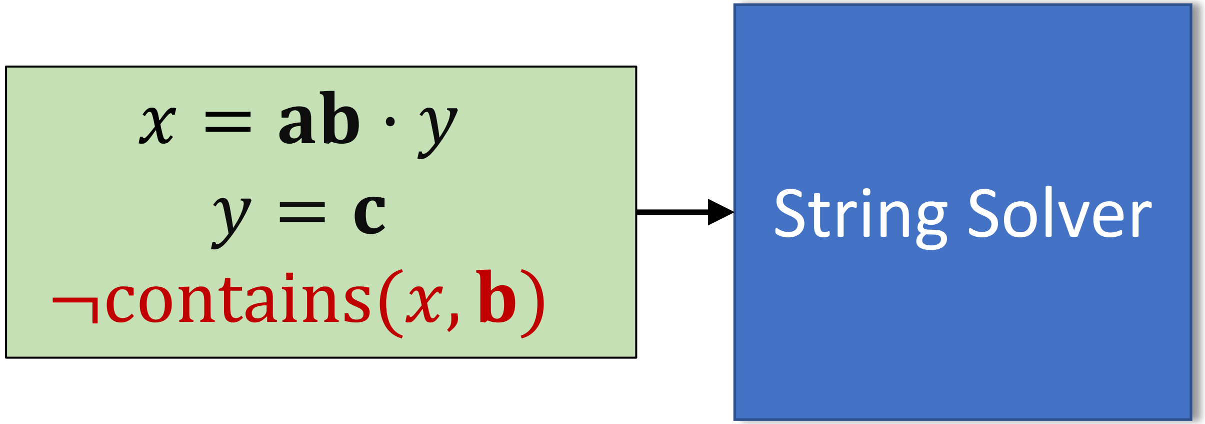
String Solver

$$\underbrace{x \neq \mathbf{abc} \cdot y \ \lor \ x \notin \mathbf{b} \cdot \Sigma^*}_{\text{Conflict Clause}}$$

Symbolic approach to RE constraint solving

- Yields a decision procedure over a reasonable fragment
- Gives rise to an incremental RE subsolver

Extended Theory of Strings [Reynolds et al., CAV'17]



$x = \mathbf{ab} \cdot y$

$y = \mathbf{c}$

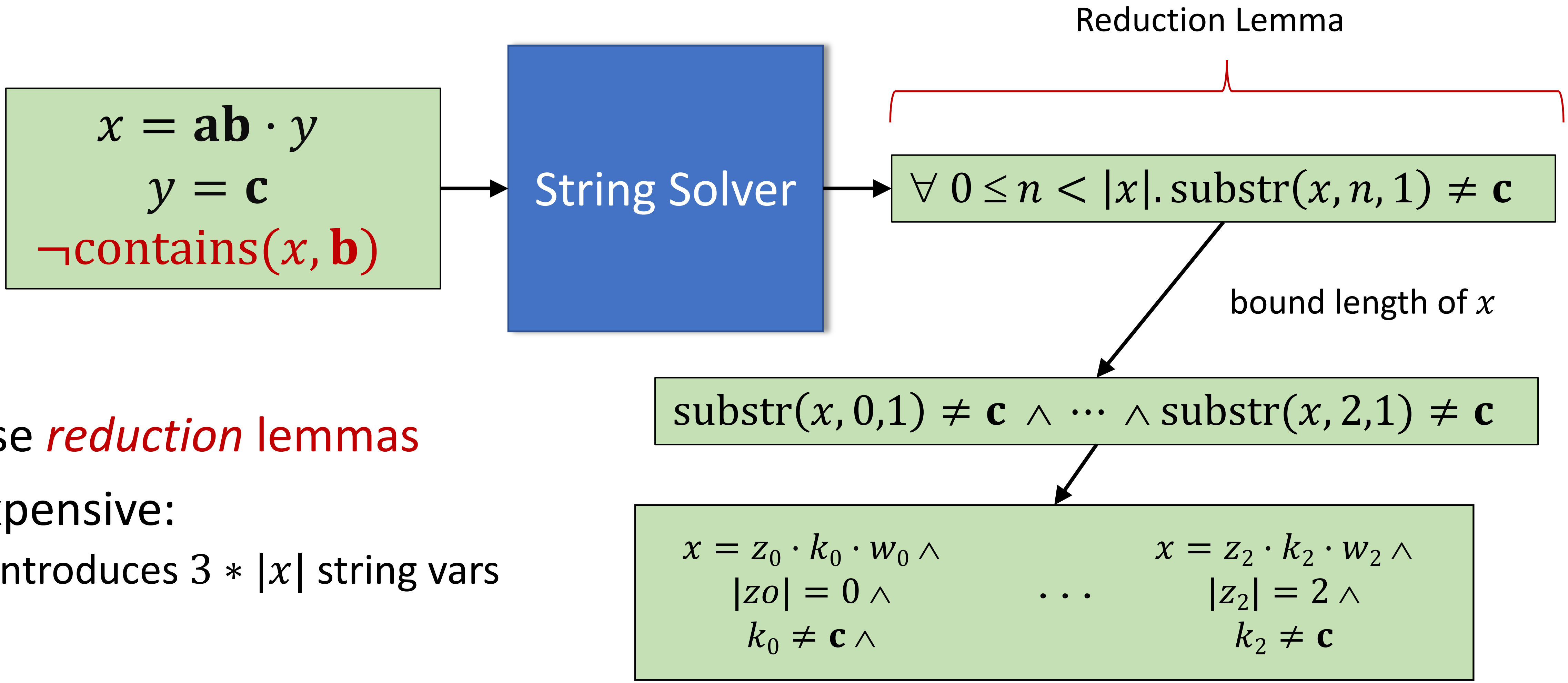
$\neg\text{contains}(x, \mathbf{b})$

String Solver

Support for *extended* string functions commonly used in applications

$\text{substr}(x, n, m)$	substring of x at position n of length at most m
$\text{contains}(x, y)$	true if string x contains substring y
$\text{indexof}(x, y, n)$	position of string y in string x, starting from position n
$\text{replace}(x, y, z)$	result of replacing first occurrence of y in x by z

Extended Theory of Strings [Reynolds et al., CAV’17]



$x = \mathbf{ab} \cdot y$

$y = \mathbf{c}$

$\neg\text{contains}(x, \mathbf{b})$

String Solver

Reduction Lemma

$\forall\ 0 \le n < |x|.\ \text{substr}(x, n, 1) \neq \mathbf{c}$

bound length of x

$\text{substr}(x, 0{,}1) \neq \mathbf{c} \wedge \cdots \wedge \text{substr}(x, 2{,}1) \neq \mathbf{c}$

$x = z_0 \cdot k_0 \cdot w_0 \wedge$
$|zo| = 0 \wedge$
$k_0 \neq \mathbf{c} \wedge$

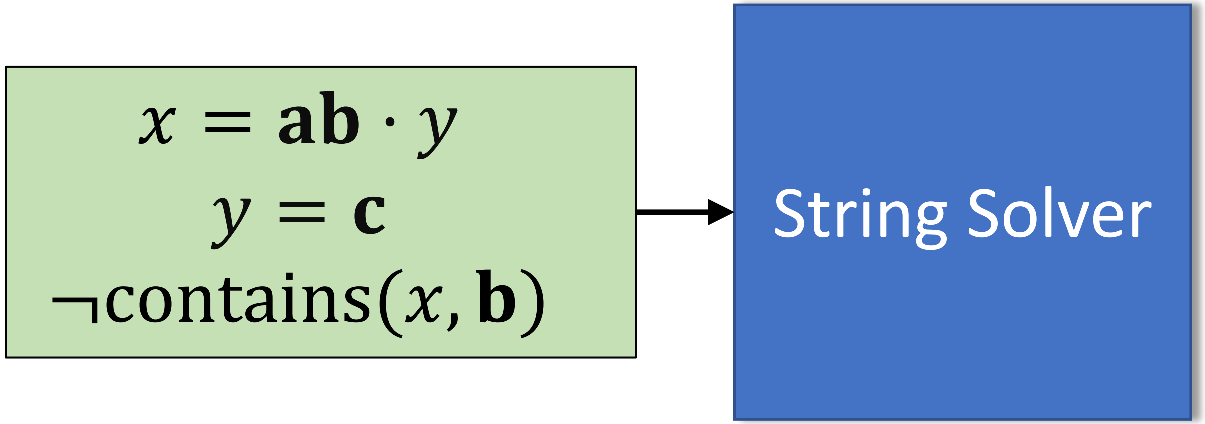
$\cdots$

$x = z_2 \cdot k_2 \cdot w_2 \wedge$
$|z_2| = 2 \wedge$
$k_2 \neq \mathbf{c}$

- Use *reduction* lemmas
- Expensive:

  Introduces $3 * |x|$ string vars

Extended Theory of Strings [Reynolds et al., CAV'17]



$$x = \mathbf{ab} \cdot y$$
$$y = \mathbf{c}$$
$$\neg\text{contains}(x, \mathbf{b})$$

String Solver

Context-dependent simplification crucial for performance

- $x = \mathbf{ab} \cdot y,\ y = \mathbf{c} \ \models\ x = \mathbf{abc}$
- $\neg\text{contains}(x, \mathbf{b}) \ \rightarrow\ \neg\text{contains}(\mathbf{abc}, \mathbf{b}) \ \rightarrow\ \neg\top \ \rightarrow\ \bot$

Proof Certificates for Unsat String Constraints

- Part of general effort to make cvc5 fully proof producing [Barbosa et al., CACM’23]
- Covers great majority of the system
- Several proof granularity levels
- Evaluated on many SMT-LIB theories, including strings [Barbosa et al., IJCAR’22]
- Fine-grained proofs for rewrites, for strings [Noetzli et al., FMCAD’22]

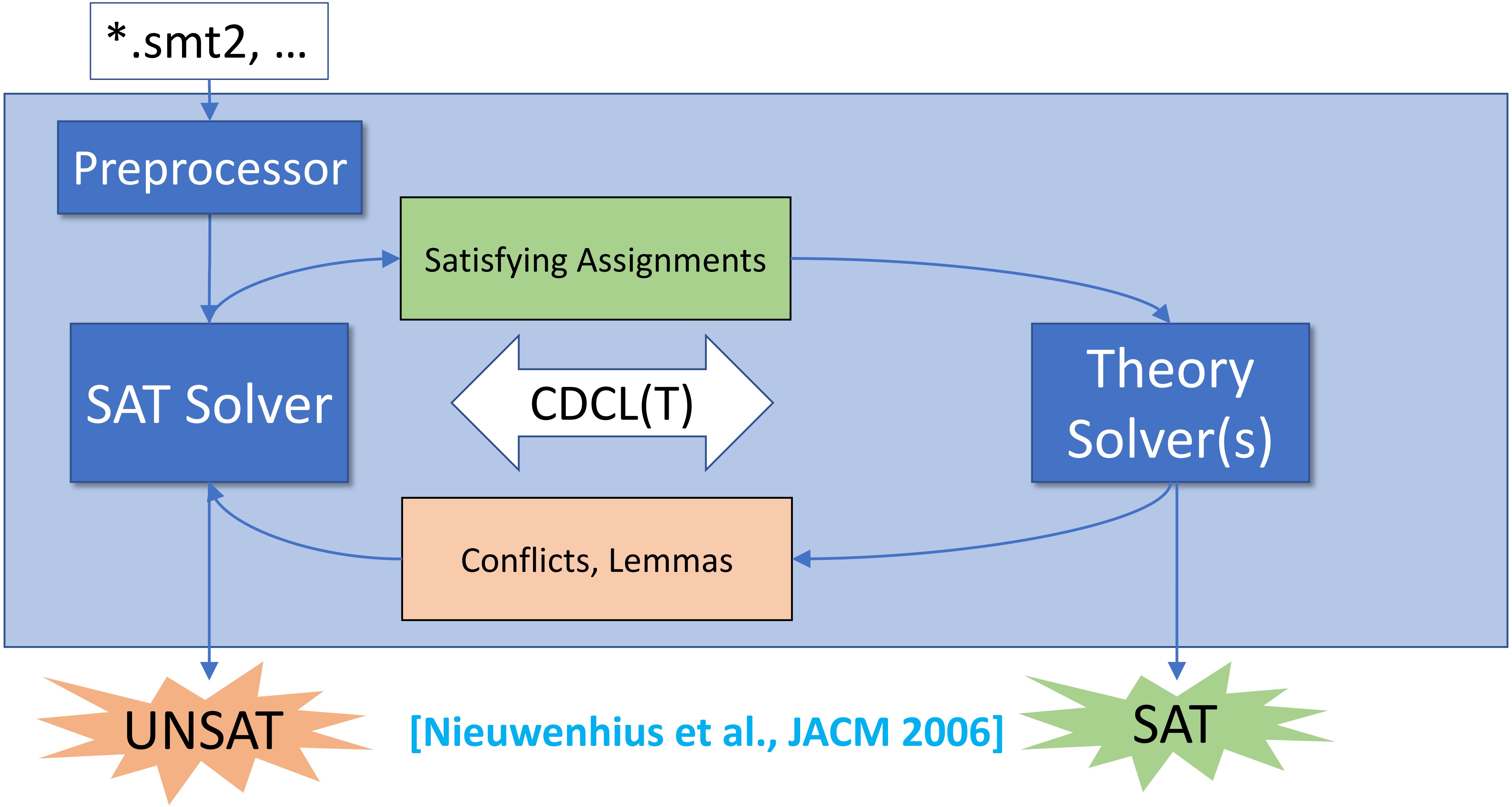
Recent Developments for Theory of Strings

- Context-dependent simplifications
    - Use aggressive rewriting [Reynolds et al., CAV 2019]
    - Applied eagerly [Noetzli et al., CAV 2022]

- Reduction lemmas
    - Leverage string-to-code point (`code`) conversion [Reynolds et al., IJCAR 2020]
    - Improved encodings [Reynolds et al., FMCAD 2020]
    - Applied lazily based on model [Noetzli et al., CAV 2022]

SMT Solvers Architecture

Architecture of most SMT solvers



*.smt2, …

Preprocessor

Satisfying Assignments

SAT Solver

CDCL(T)

Theory Solver(s)

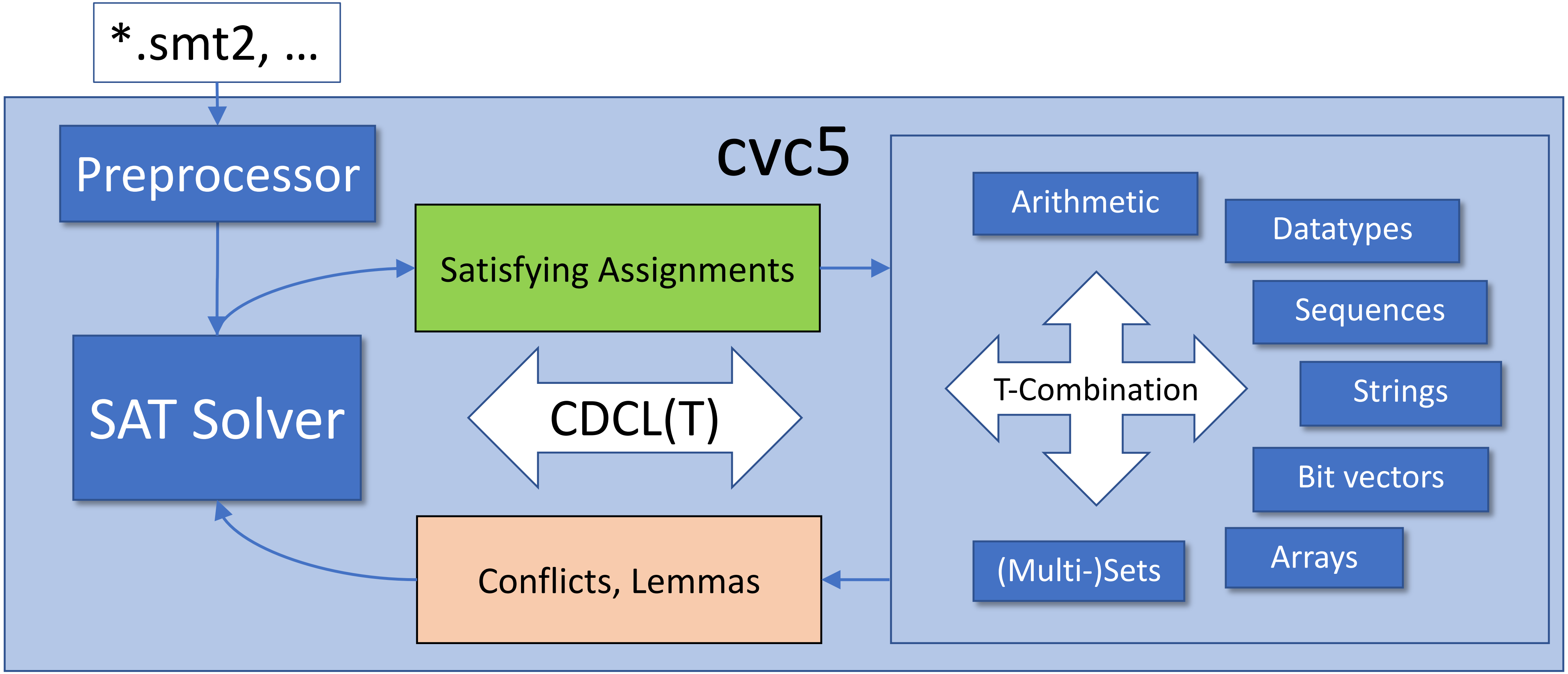
Conflicts, Lemmas

UNSAT

[Nieuwenhius et al., JACM 2006]

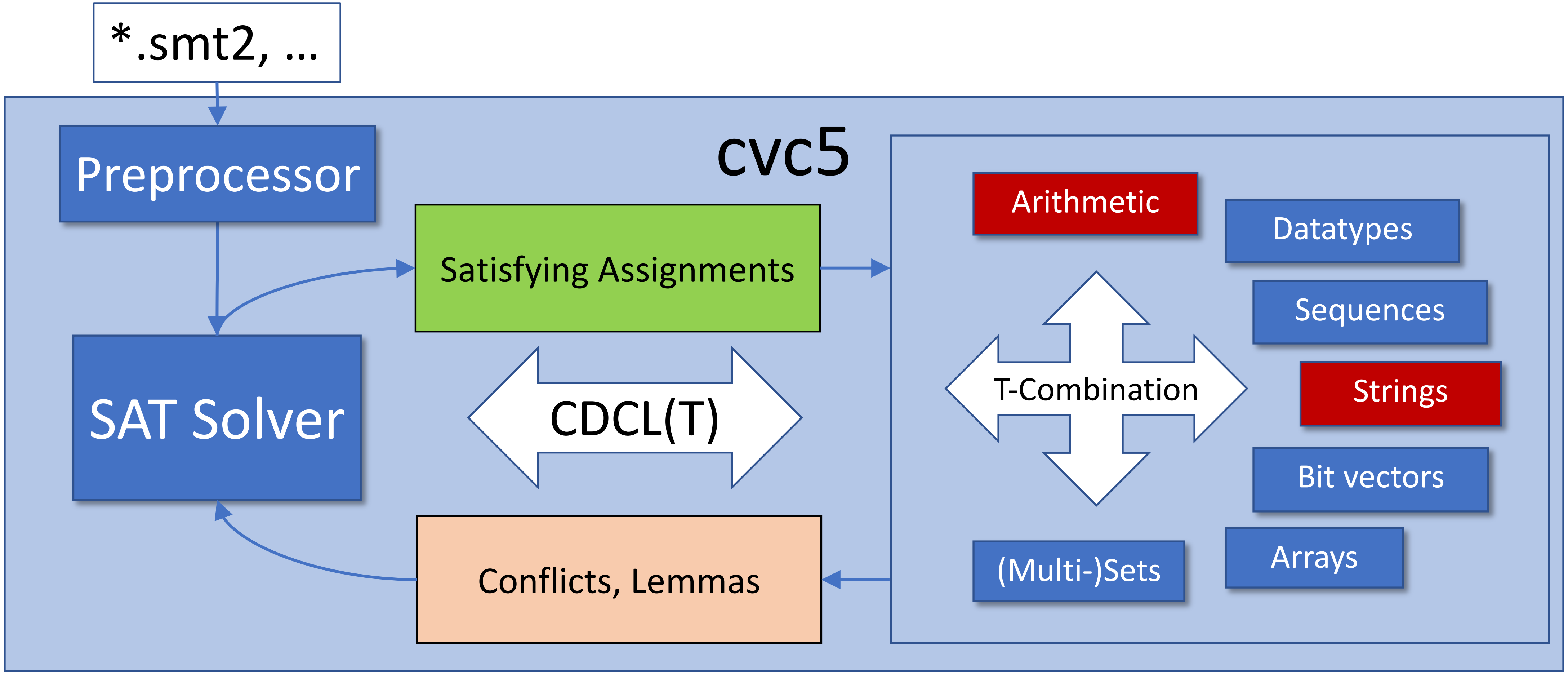
SAT

Architecture of cvc5



Centralized methods (Nelson-Oppen, polite) for combining theories

Architecture of cvc5



Focus of this talk: theory of strings and regular expressions

Theory of Strings + Linear Arithmetic (T_{SLIA})

Sorts:

- Integers Int
- Strings String, interpreted as Σ^* for finite alphabet Σ

Terms:

String variables: x, y, z, u, w
Integer variables: i, j, k
String constants: ε, $\mathbf{abc}$, $\mathbf{AcBAA}$, $\mathbf{http}$
String concatenation: $x \cdot \mathbf{abc}$, $x \cdot y \cdot z \cdot w$
String length: $|x|$

Formulas:

- Equalities and disequalities between string terms
- Linear arithmetic constraints: $|x| + 4 > |y|$

Example: $x \cdot \mathbf{a} = y, \quad y \neq \mathbf{b} \cdot z, \quad |y| > |x| + 2$

Although decidability is unknown, many problems can be solved efficiently in practice

CDCL(T) String Solvers

Cooperation between:

SAT Solver

Arithmetic Solver

String Solver

CDCL(T) String Solvers

$$x = \mathbf{ab} \cdot z$$

$$|x| + |y| \leq 5$$

$$\mathbf{abcd} \cdot x = y \vee |x| > 5$$

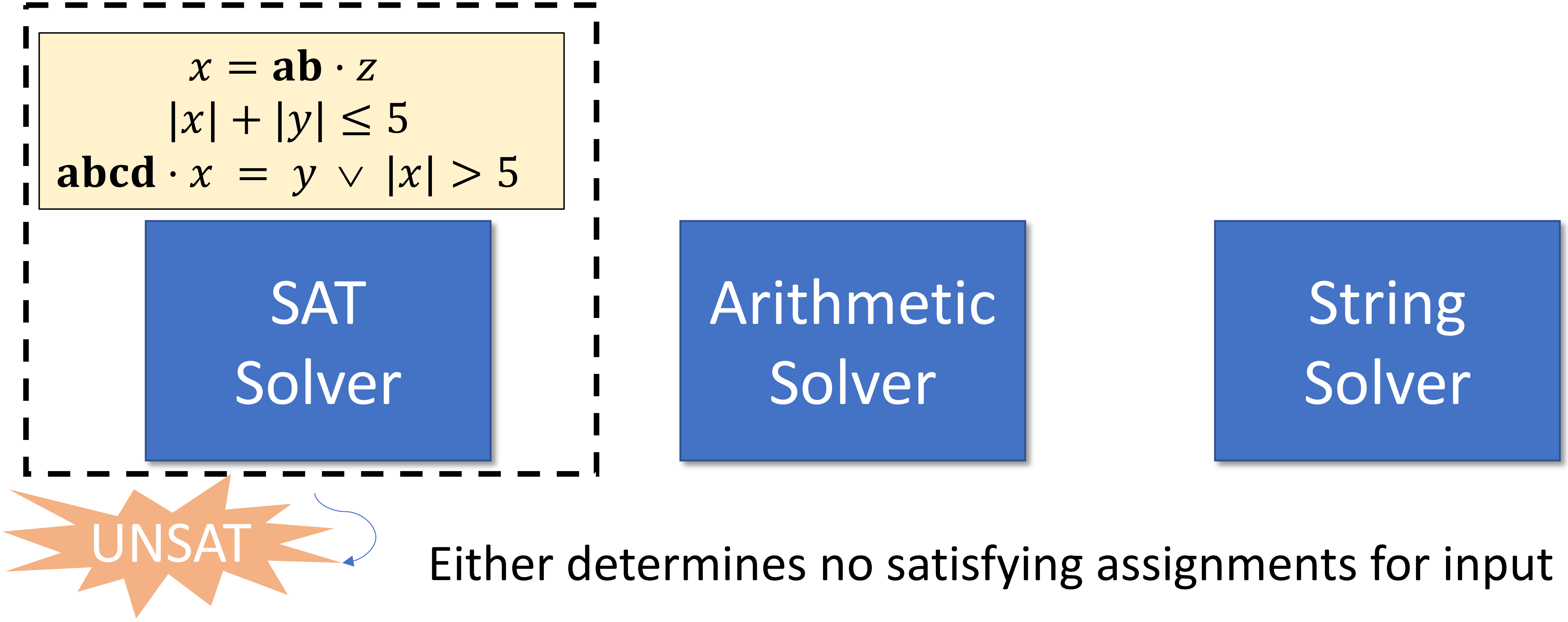
Set of T_{SLIA}-formulas in clausal normal form (CNF)

SAT Solver

Arithmetic Solver

String Solver

CDCL(T) String Solvers



$$x = \mathbf{ab} \cdot z$$

$$|x| + |y| \leq 5$$

$$\mathbf{abcd} \cdot x = y \vee |x| > 5$$

Either determines no satisfying assignments for input exist …

CDCL(T) String Solvers

$$x = \mathbf{ab} \cdot z$$
$$|x| + |y| \leq 5$$
$$\mathbf{abcd} \cdot x = y \vee |x| > 5$$

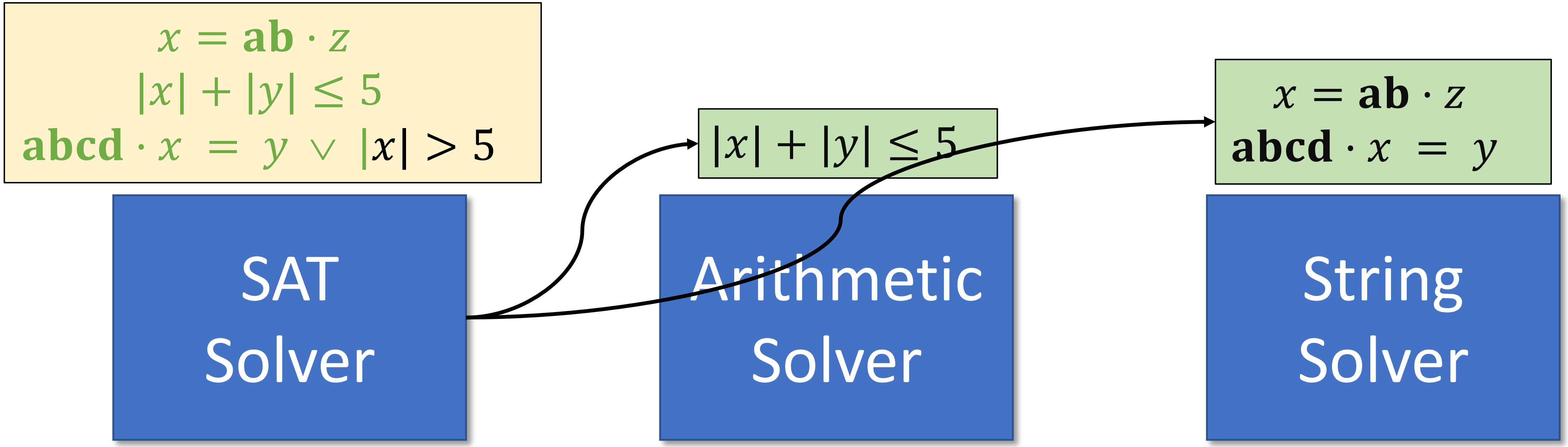
SAT
Solver

Arithmetic
Solver

String
Solver

… or returns a propositionally satisfying assignment

CDCL(T) String Solvers



$x = \mathbf{ab} \cdot z$

$|x| + |y| \leq 5$

$\mathbf{abcd} \cdot x = y \vee |x| > 5$

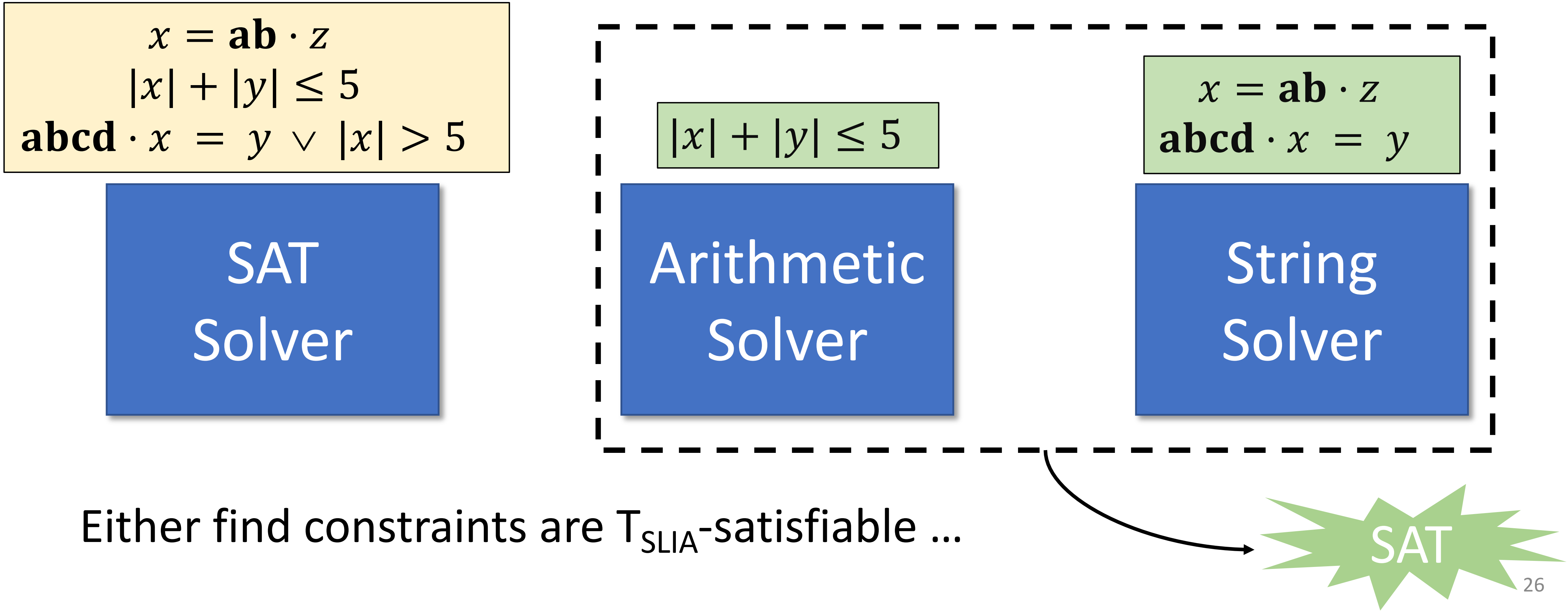
$|x| + |y| \leq 5$

$x = \mathbf{ab} \cdot z$

$\mathbf{abcd} \cdot x = y$

⇒ Constraints distributed to arithmetic and string solvers

CDCL(T) String Solvers



$x = \mathbf{ab} \cdot z$

$|x| + |y| \leq 5$

$\mathbf{abcd} \cdot x = y \vee |x| > 5$

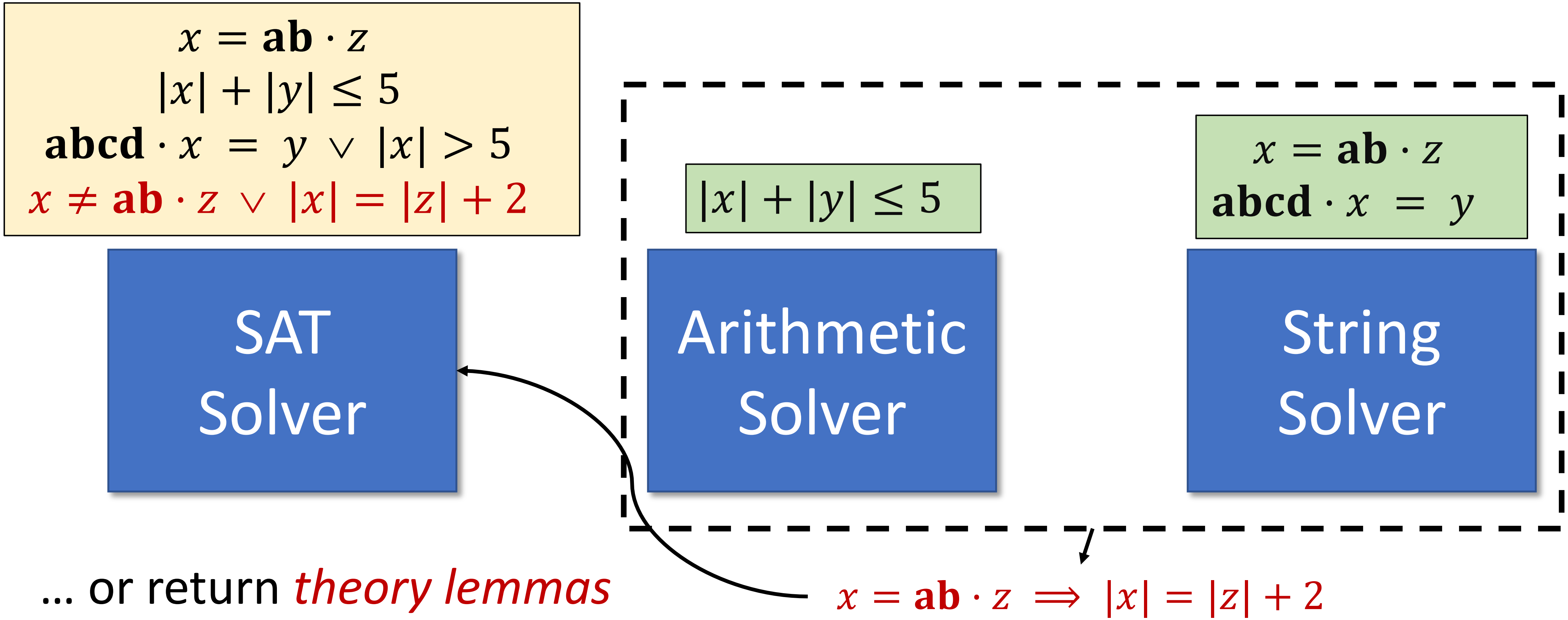
$|x| + |y| \leq 5$

$x = \mathbf{ab} \cdot z$

$\mathbf{abcd} \cdot x = y$

Either find constraints are T_{SLIA}-satisfiable …

CDCL(T) String Solvers



$x = \mathbf{ab} \cdot z$

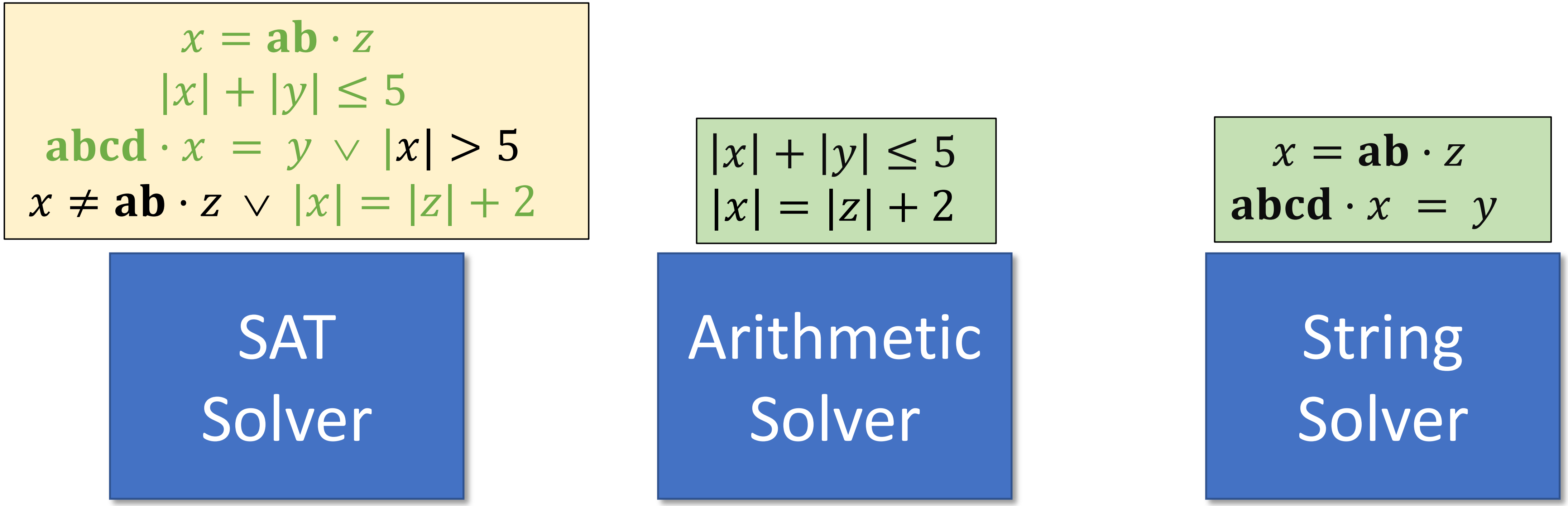
$|x| + |y| \leq 5$

$\mathbf{abcd} \cdot x = y \vee |x| > 5$

$x \neq \mathbf{ab} \cdot z \vee |x| = |z| + 2$

… or return *theory lemmas*
(valid $T_{\text{LIA}}/T_{\text{S}}$-formulas) to SAT solver …

CDCL(T) String Solvers



$x = \mathbf{ab} \cdot z$
$|x| + |y| \leq 5$
$\mathbf{abcd} \cdot x = y \lor |x| > 5$
$x \neq \mathbf{ab} \cdot z \lor |x| = |z| + 2$

SAT Solver

$|x| + |y| \leq 5$
$|x| = |z| + 2$

Arithmetic Solver

$x = \mathbf{ab} \cdot z$
$\mathbf{abcd} \cdot x = y$

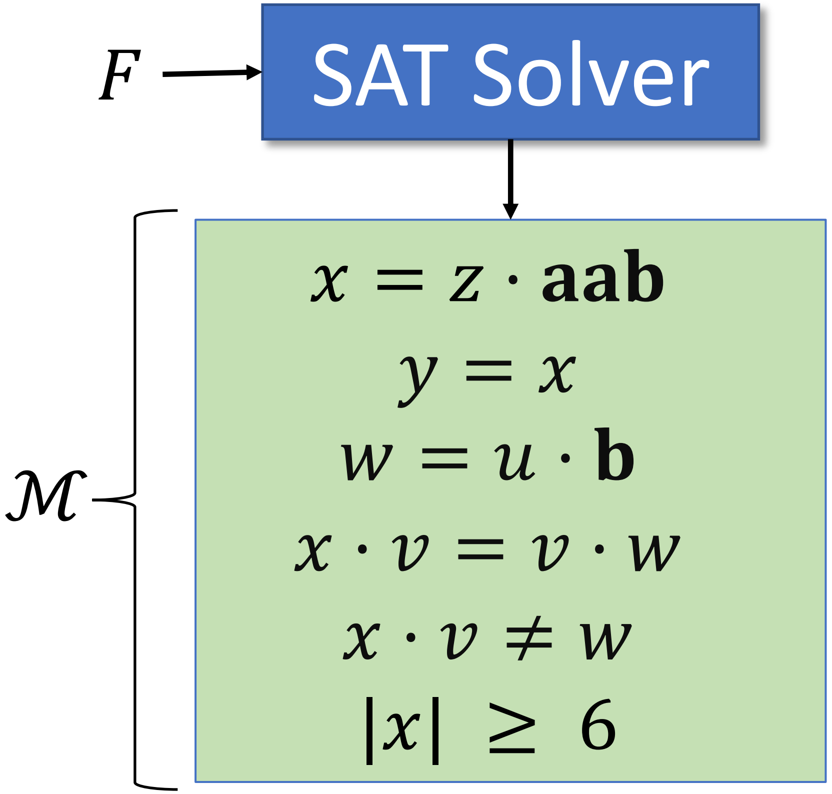
String Solver

… and repeat

A Theory Solver for Strings

[Liang, Reynolds, Deters, Tinelli and Barrett, CAV 14]

Solving String Constraints



F

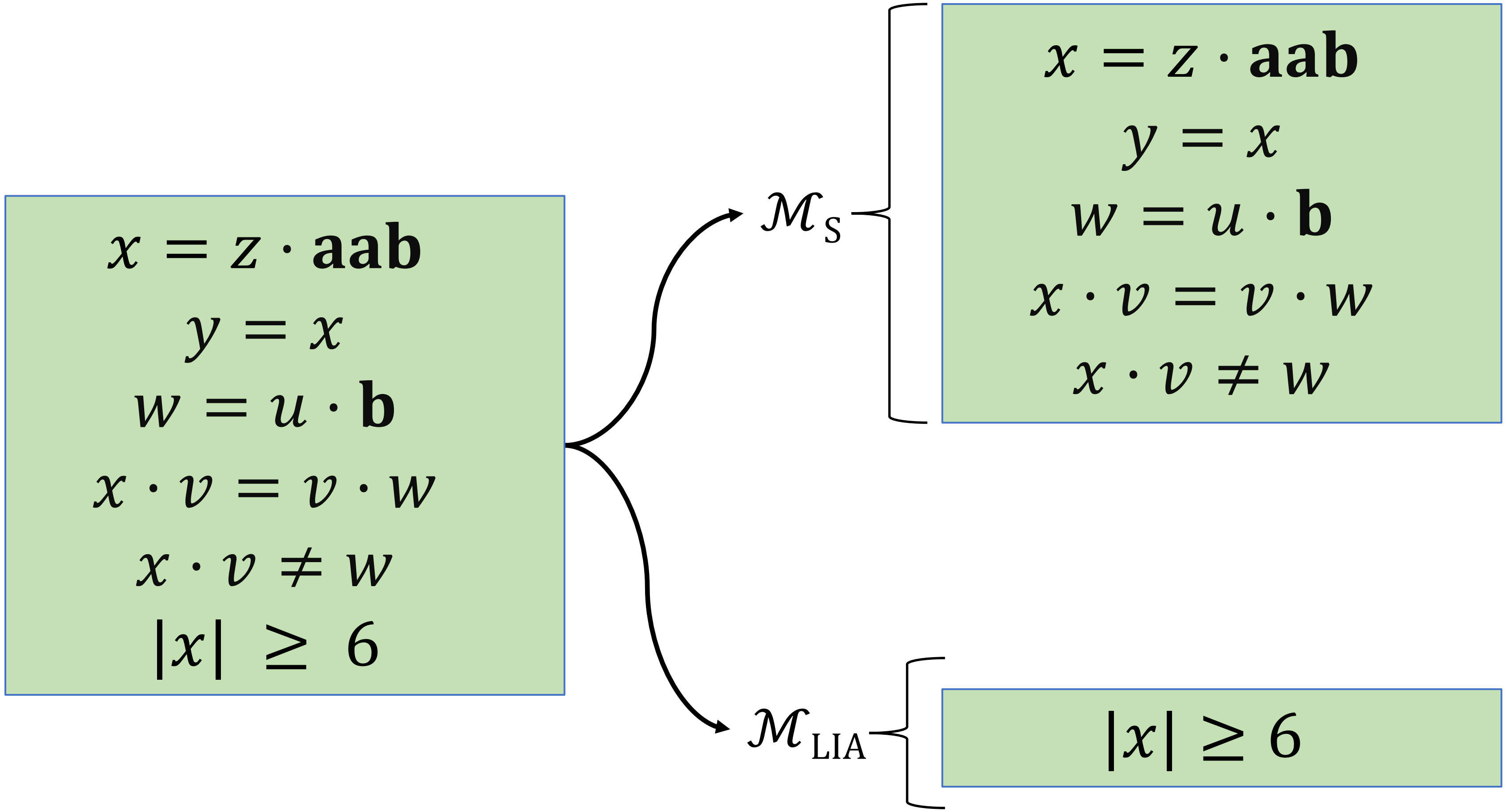
SAT Solver

$\mathcal{M}$

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$
$$|x| \geq 6$$

where $\mathcal{M} \vDash_{\mathrm{p}} F$

Solving String Constraints



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$
$$|x| \geq 6$$

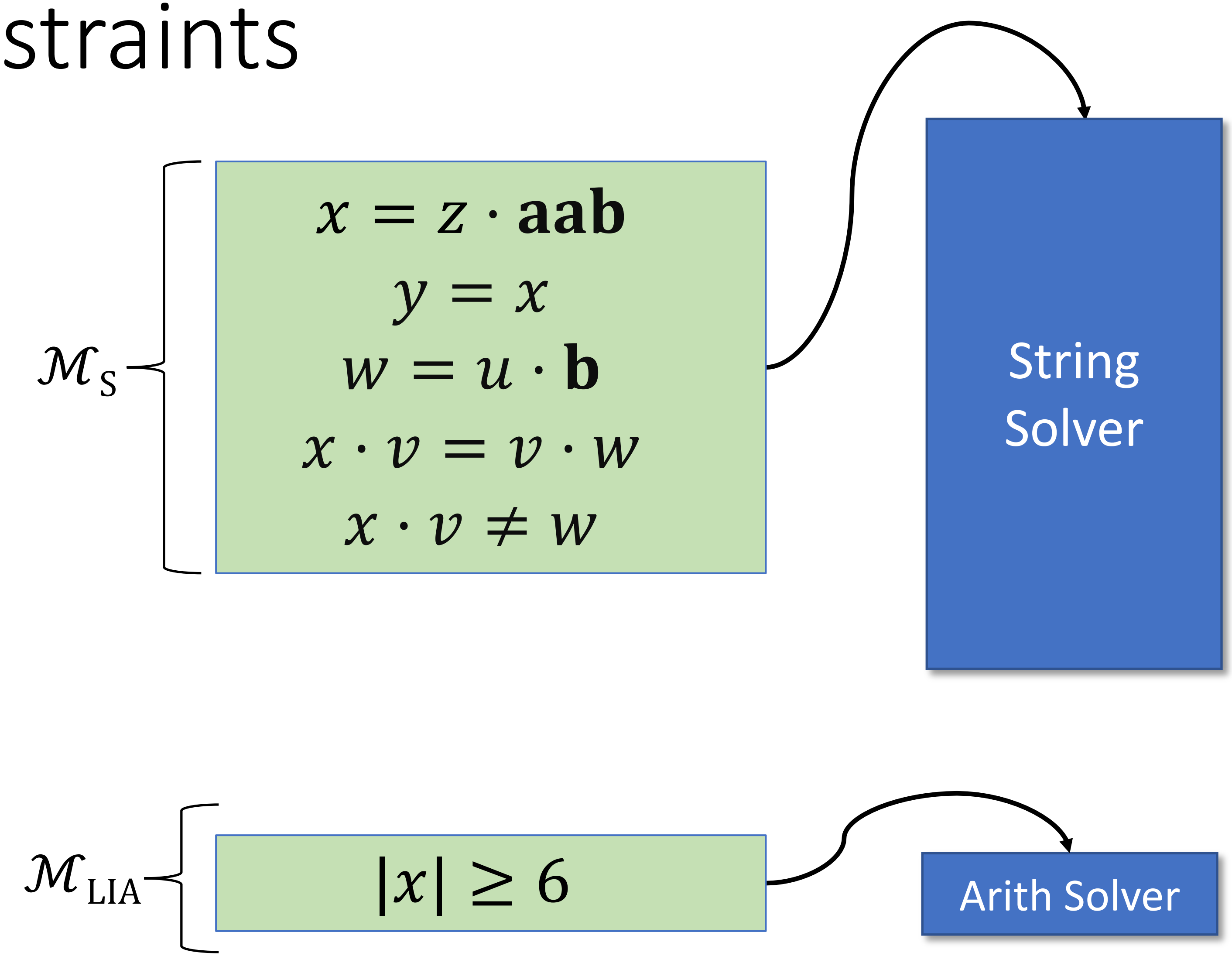
$\mathcal{M}_{\mathrm{S}}$

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

$\mathcal{M}_{\mathrm{LIA}}$

$$|x| \geq 6$$

Solving String Constraints



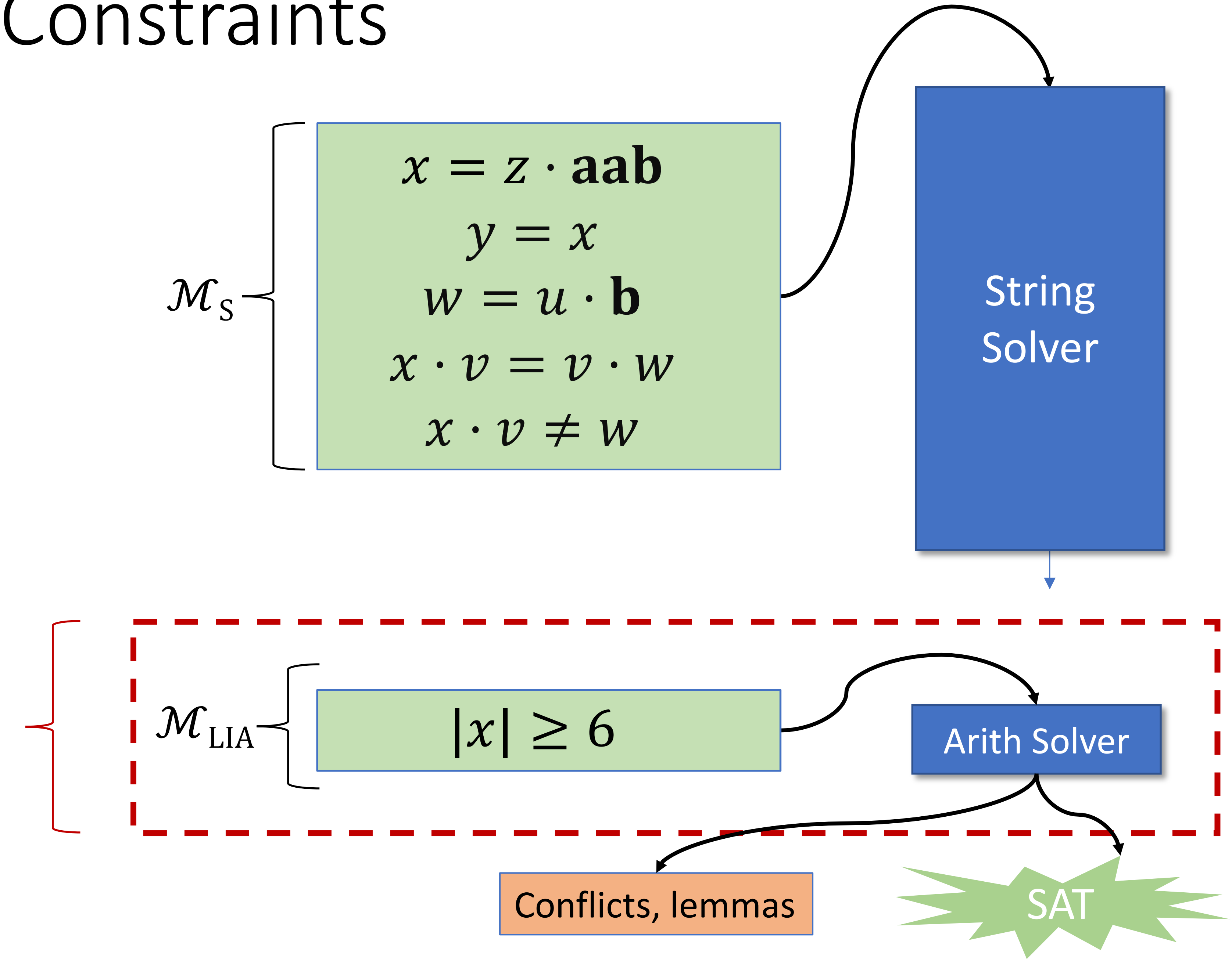
$$\mathcal{M}_{\mathrm{S}} \left\{ \begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array} \right.$$

String Solver

$$\mathcal{M}_{\mathrm{LIA}} \left\{ |x| \geq 6 \right.$$

Arith Solver

Solving String Constraints



$\mathcal{M}_{\mathrm{S}}$

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

String Solver

Theory Solver for Linear Integer Arithmetic (Simplex)

$\mathcal{M}_{\mathrm{LIA}}$

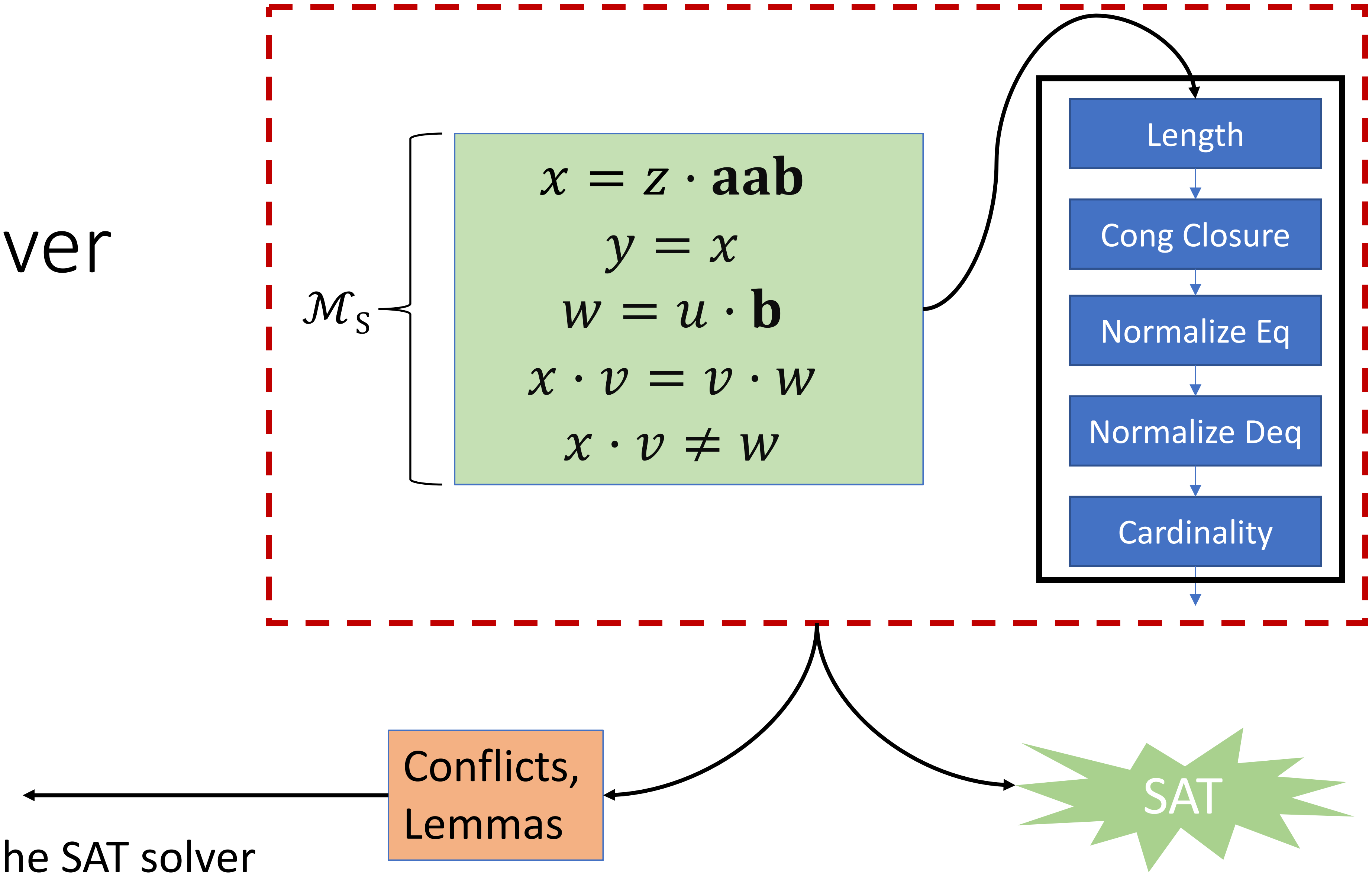
$$|x| \geq 6$$

Arith Solver

Conflicts, lemmas

SAT

Theory Solver for Strings



$$\mathcal{M}_S \left\{ \begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array} \right.$$

Length

Cong Closure

Normalize Eq

Normalize Deq

Cardinality

Conflicts, Lemmas

back to the SAT solver

SAT

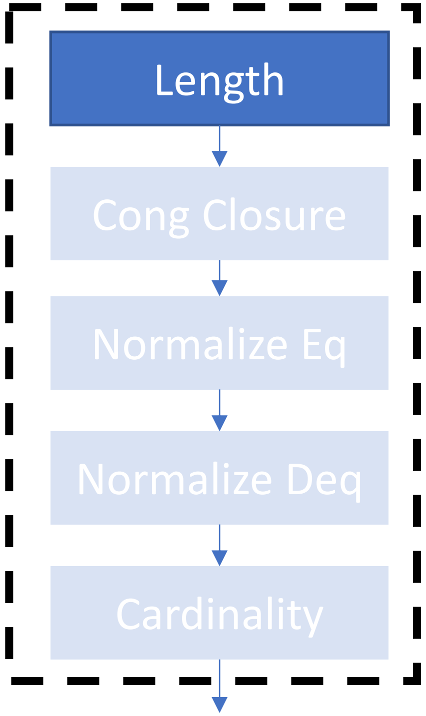
String Theory Solver Inference Strategy

1. Elaborate length constraints
2. Check for equality conflicts (compute congruence closure)
3. Normalize string equalities
4. Normalize string disequalities
5. Check cardinality constraints

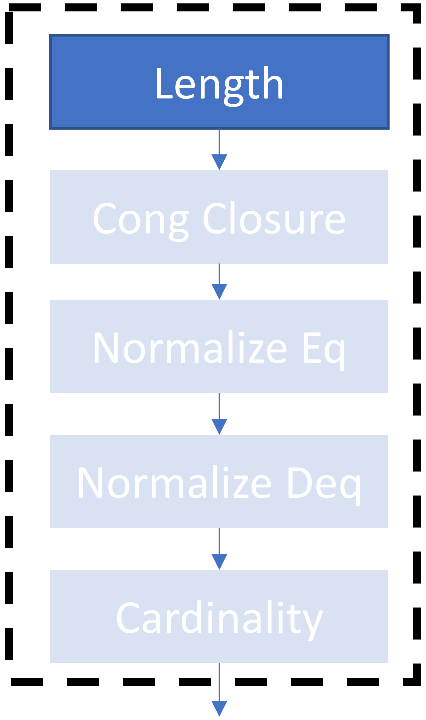
- Each step may add lemma or a conflict
- If no step adds a lemma or conflict, the current constraint set ($\mathcal{M}_S \cup \mathcal{M}_S$) is sat

1. Elaborate Length Constraints

$$\mathcal{M}_S \left\{ \boxed{\begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array}} \right.$$



1. Elaborate Length Constraints



Length

Cong Closure

Normalize Eq

Normalize Deq

Cardinality

$$\mathcal{M}_S \left\{ \boxed{\begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array}} \right.$$

- For each term of type string in $\mathcal{M}_S$,
  add lemma providing the definition of its length:

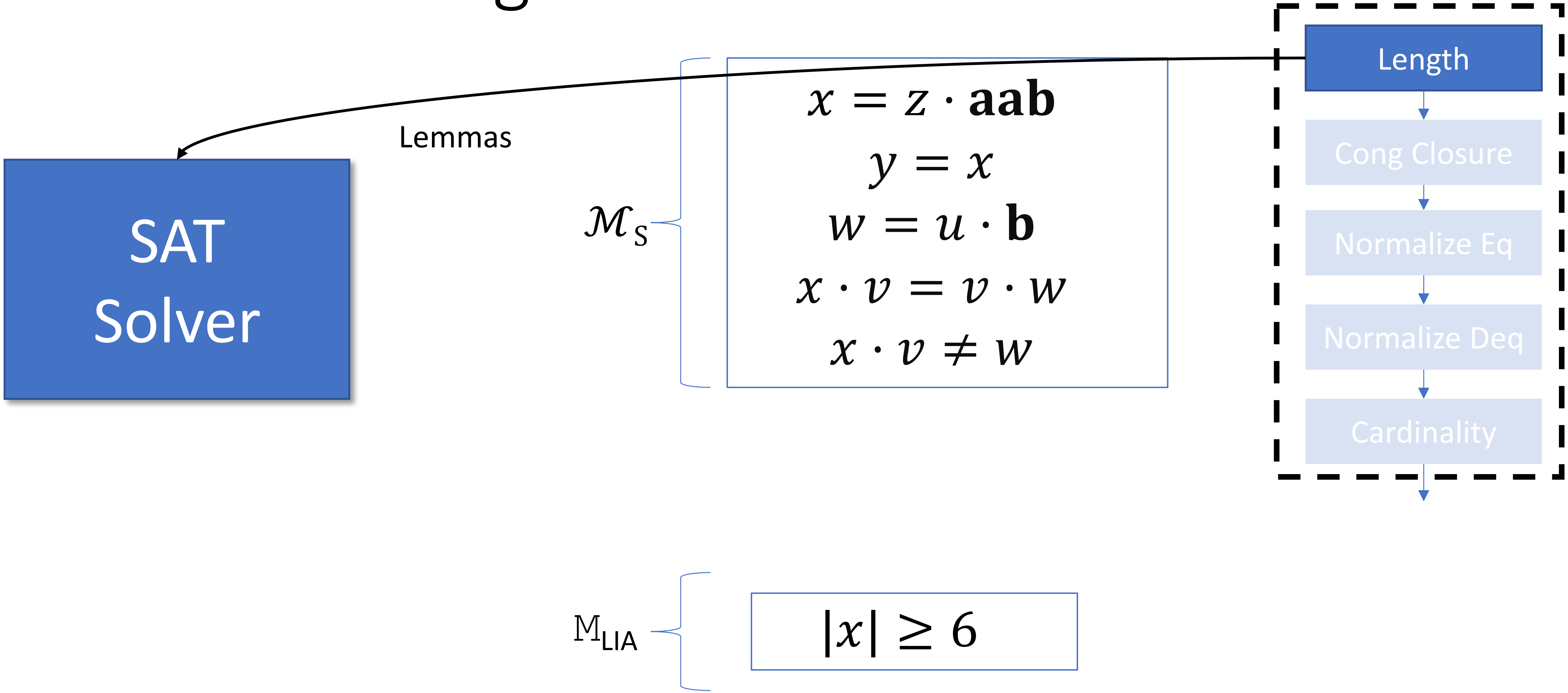
$$|\mathbf{b}| = 1 \qquad |\mathbf{aab}| = 3 \qquad |x \cdot v| = |x| + |v|$$

$$|z \cdot \mathbf{aab}| = |z| + 3 \qquad |u \cdot \mathbf{b}| = |u| + 3 \qquad |v \cdot w| = |v| + |w|$$

- For each variable of type string in $\mathcal{M}_S$,
  add an emptiness splitting lemma:

$$x = \epsilon \lor |x| \geq 1 \qquad y = \epsilon \lor |y| \geq 1 \qquad \ldots$$

1. Elaborate Length Constraints



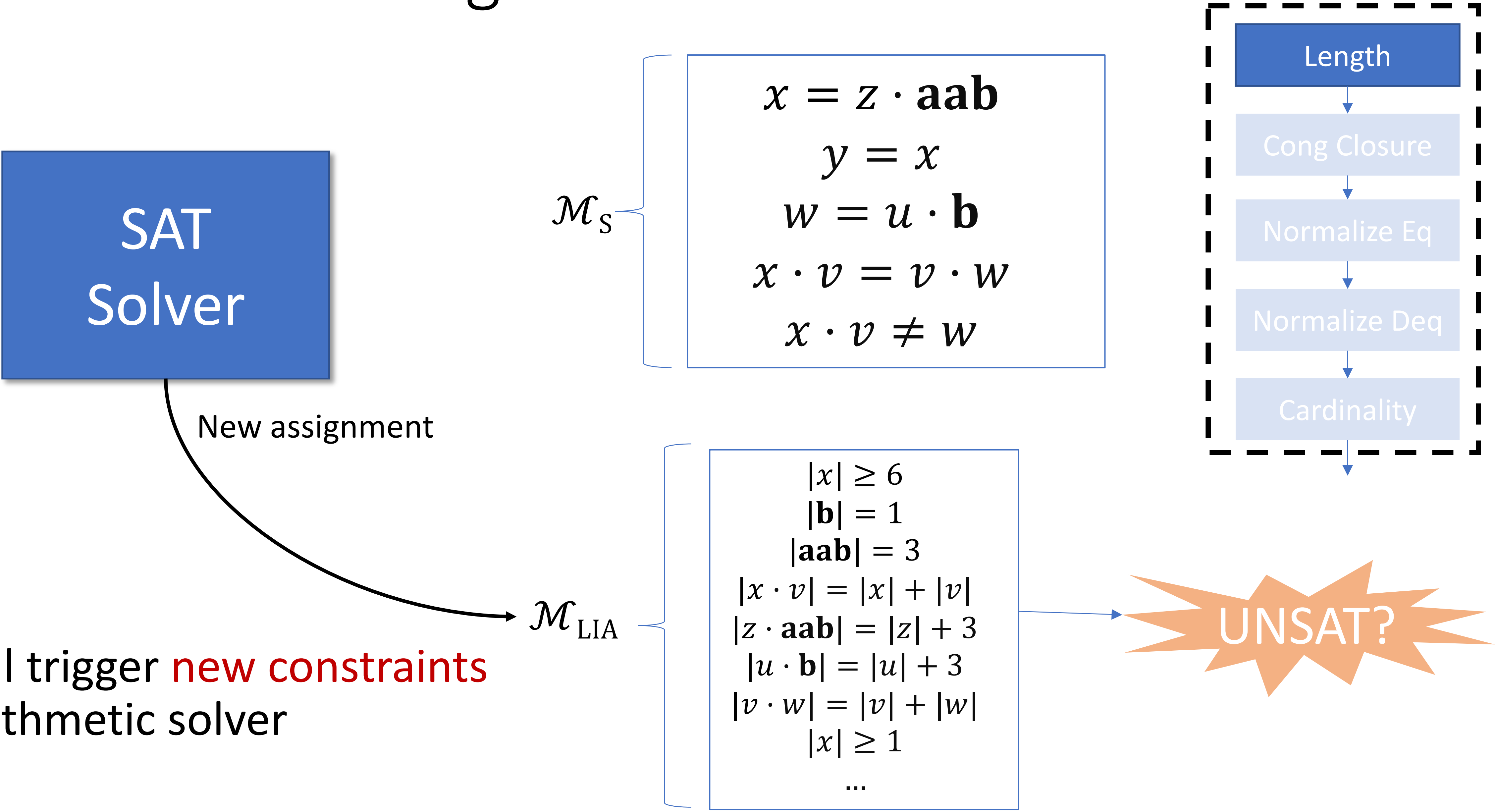
$\mathcal{M}_S$

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

$\mathrm{M}_{\mathrm{LIA}}$

$$|x| \geq 6$$

1. Elaborate Length Constraints



SAT Solver

New assignment

$\mathcal{M}_S$

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

$\mathcal{M}_{\text{LIA}}$

$$|x| \geq 6$$
$$|\mathbf{b}| = 1$$
$$|\mathbf{aab}| = 3$$
$$|x \cdot v| = |x| + |v|$$
$$|z \cdot \mathbf{aab}| = |z| + 3$$
$$|u \cdot \mathbf{b}| = |u| + 3$$
$$|v \cdot w| = |v| + |w|$$
$$|x| \geq 1$$
…

Length

Cong Closure

Normalize Eq

Normalize Deq

Cardinality

UNSAT?

… will trigger new constraints in arithmetic solver

2. Compute Congruence Closure

$$\mathcal{M}_{\mathrm{S}} \left\{ \boxed{\begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array}} \right.$$

Length

Cong Closure

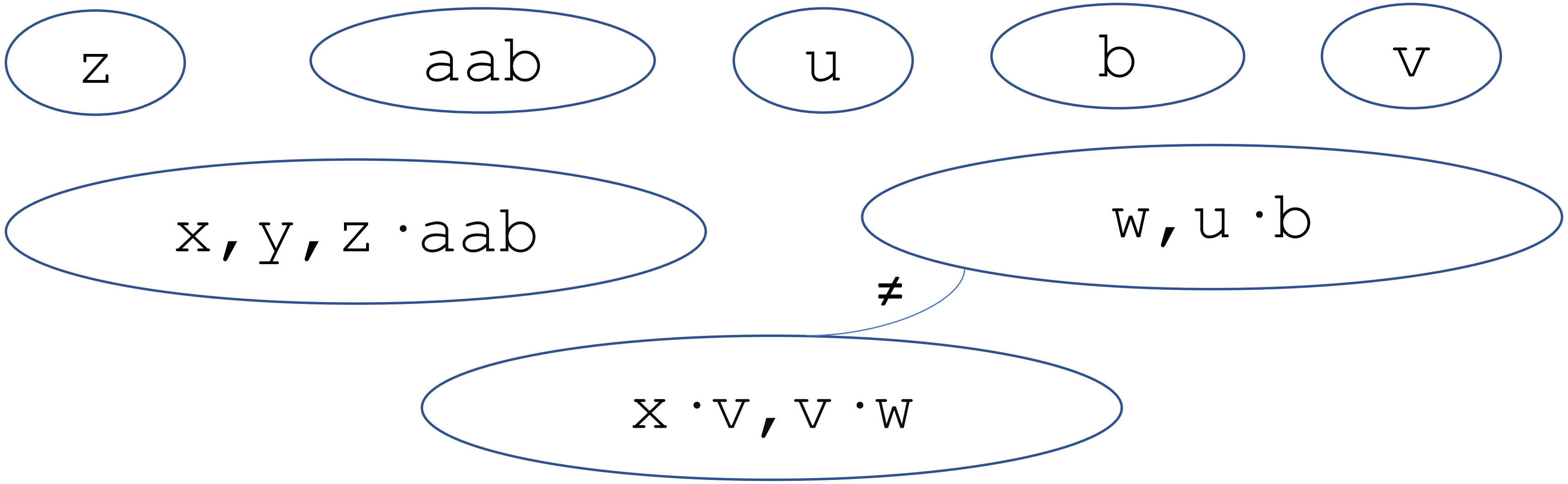
Normalize Eq

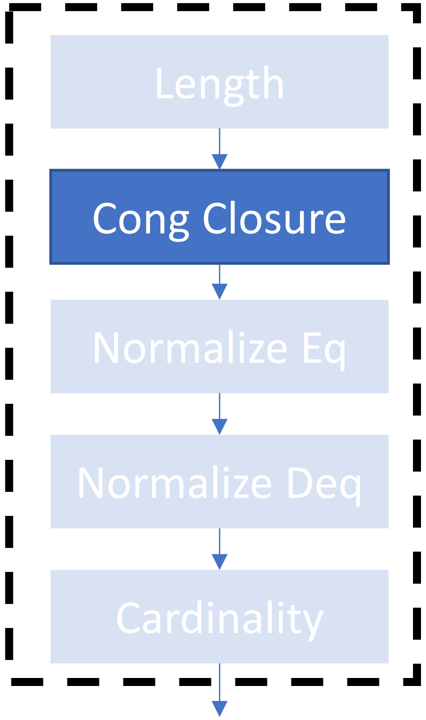
Normalize Deq

Cardinality

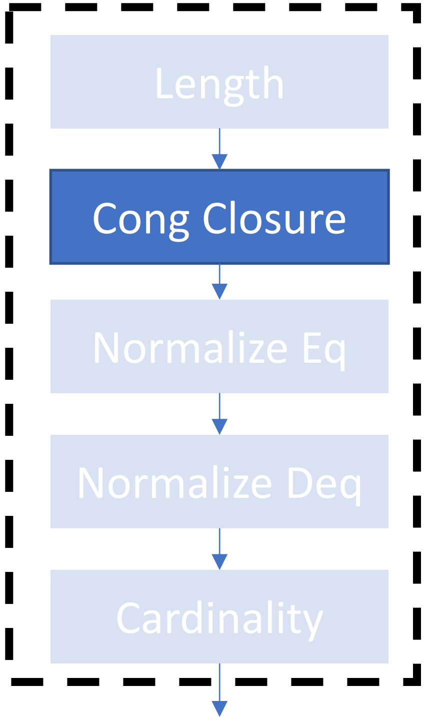
$$\mathcal{M}_S \left\{ \boxed{\begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array}} \right.$$

Group terms by equivalence classes:





$$\mathcal{M}_S \left\{ \begin{array}{c} x = z \cdot \mathbf{aab} \\ y = x \\ w = u \cdot \mathbf{b} \\ x \cdot v = v \cdot w \\ x \cdot v \neq w \end{array} \right.$$


Length

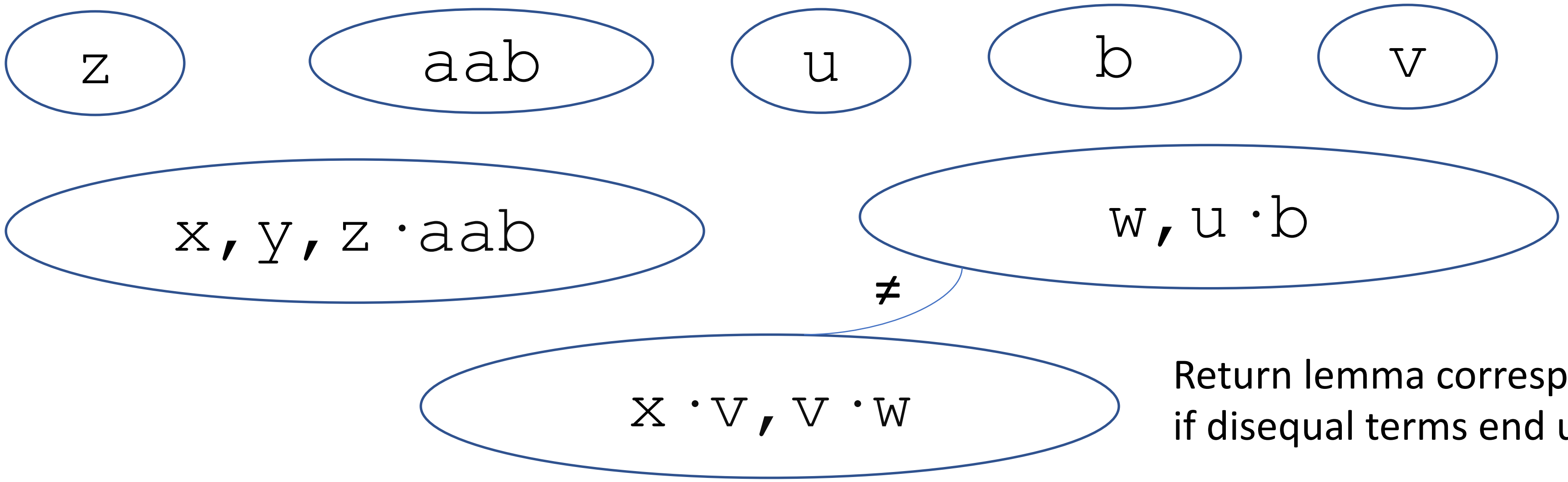
Cong Closure

Normalize Eq

Normalize Deq

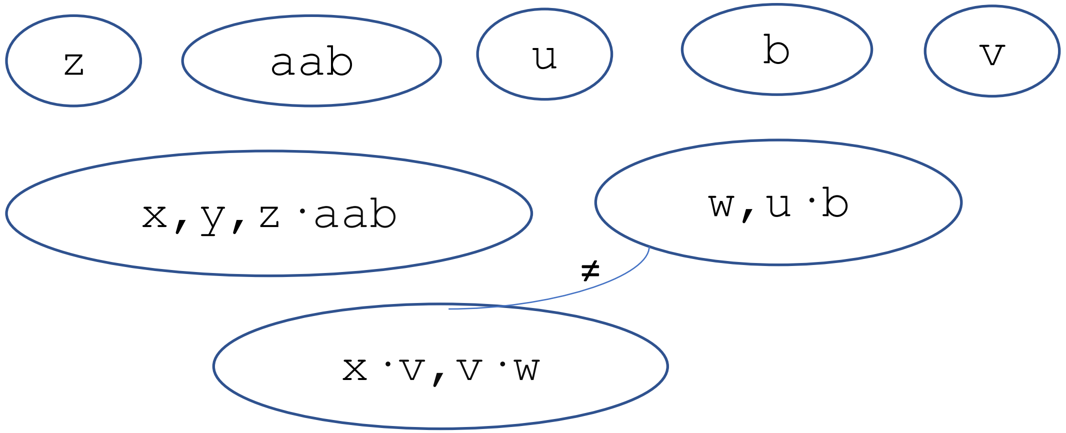
Cardinality

Group terms by equivalence classes:

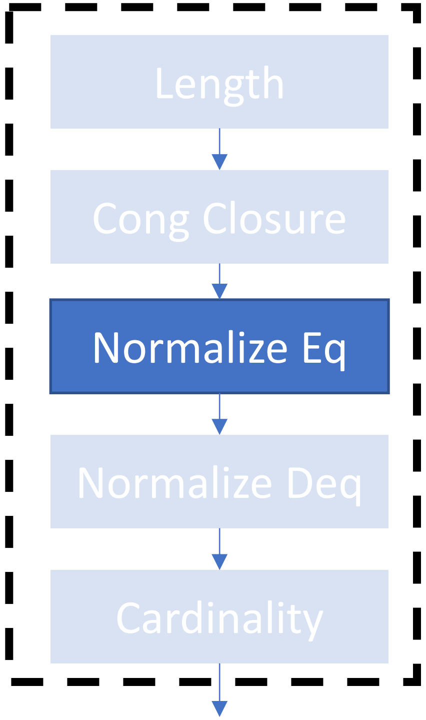


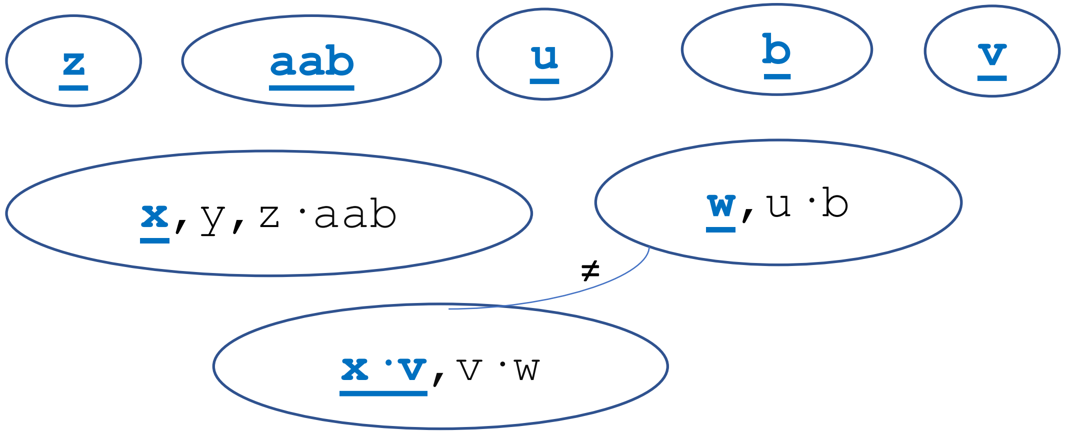
Return lemma corresponding to T_s-conflict
if disequal terms end up in the same eq class

3. Normalize Equalities

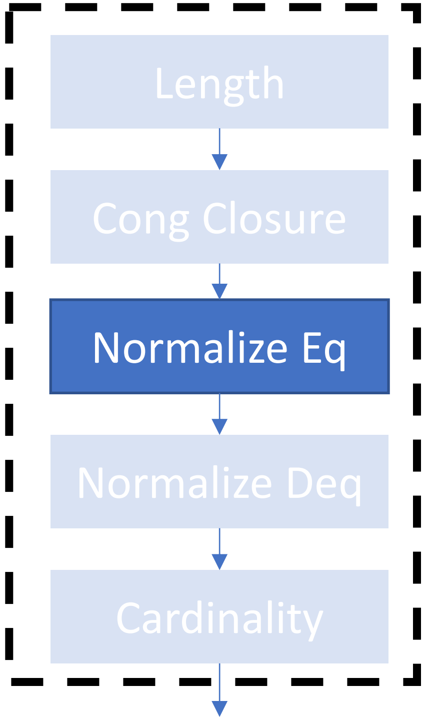


$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$





$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

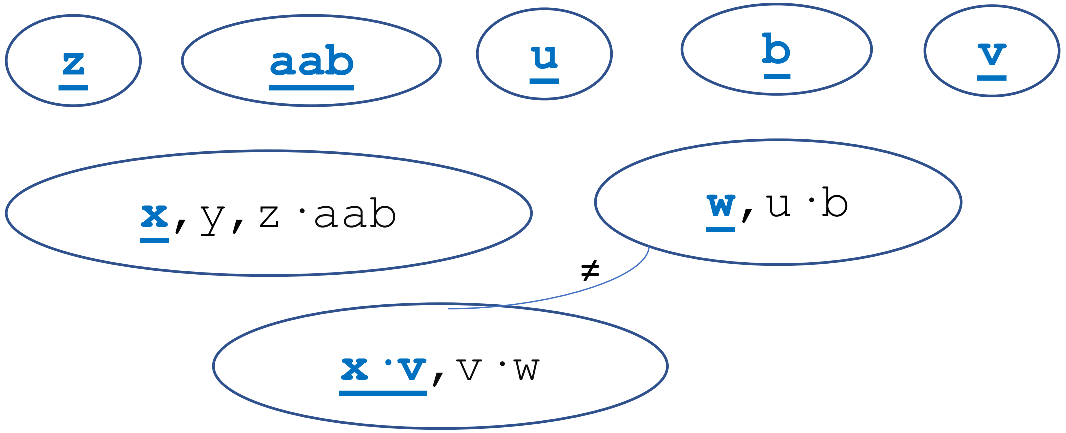


Compute *normal forms* for equivalence classes

- A normal form is a concatenation of string terms $r_1 \cdot \cdots \cdot r_n$ where each r_i is the **representative** of its equivalence class

  Restriction: string constants must be chosen as representatives
- An equivalence class can be assigned a normal form $r_1 \cdot \cdots \cdot r_n$ if:

  Each non-variable term in it can be expanded (modulo equality and rewriting) to $r_1 \cdot \cdots \cdot r_n$



z

aab

u

b

v

x, y, z·aab

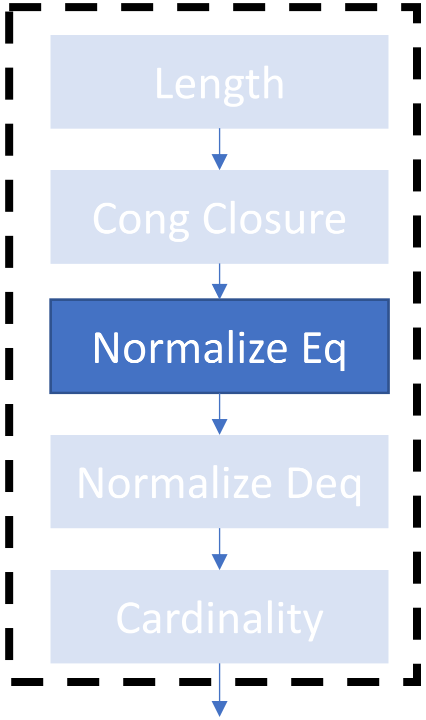
w, u·b

≠

x·v, v·w

Normal forms computed bottom-up

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$



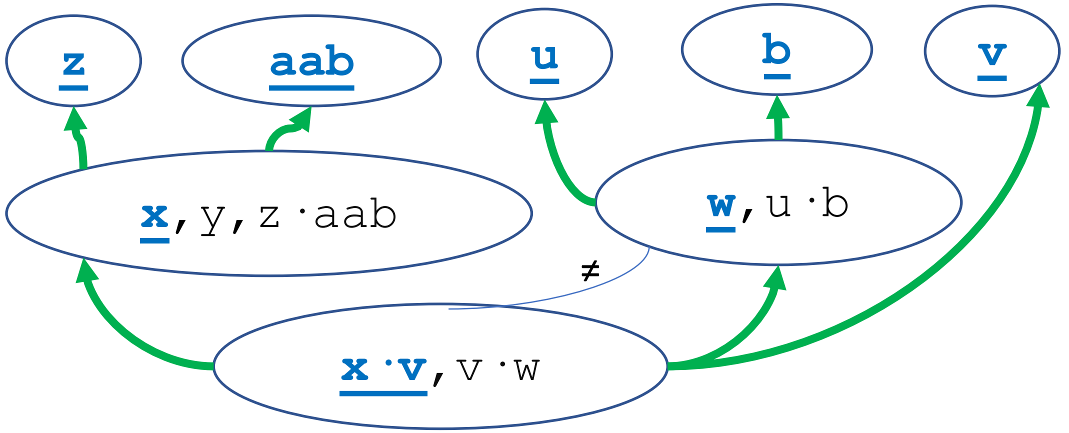
Length

Cong Closure

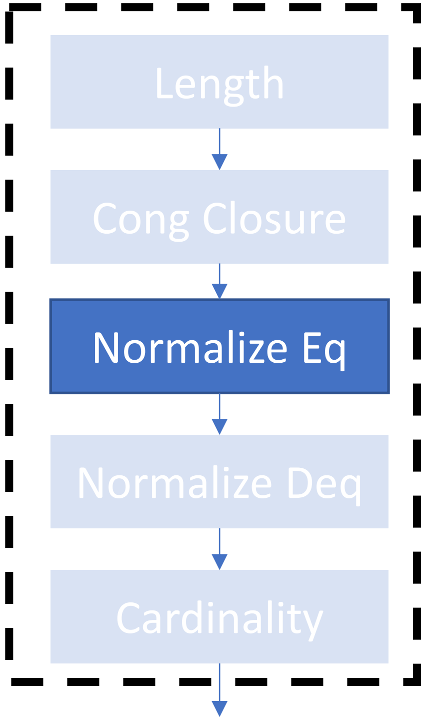
Normalize Eq

Normalize Deq

Cardinality



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$


Length

Cong Closure

Normalize Eq

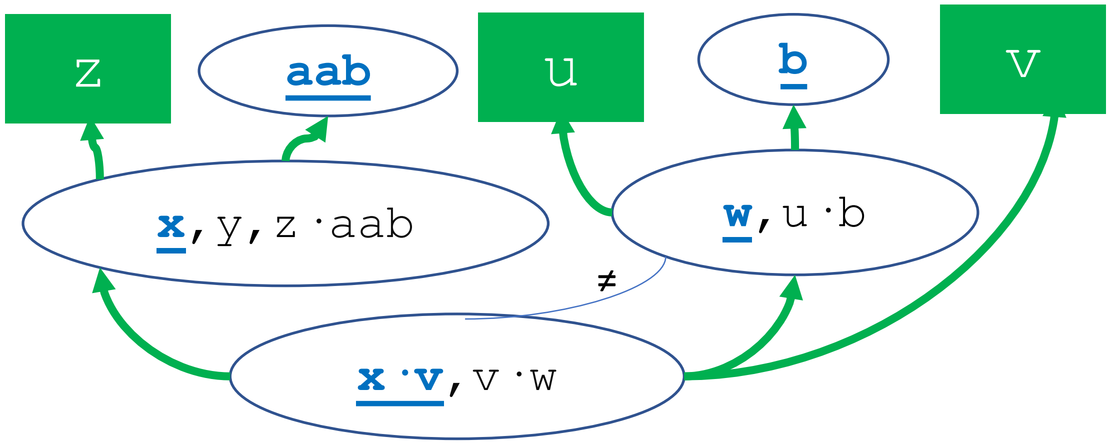
Normalize Deq

Cardinality

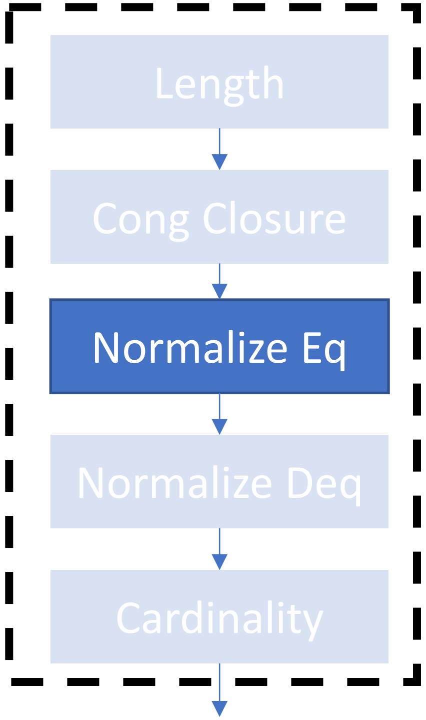
Normal forms computed bottom-up

- First, compute containment relation induced by concatenation terms

This relation is guaranteed to be acyclic due to length elaboration step (cycle ⇒ LIA-conflict)



$$
\begin{gathered}
x = z \cdot \mathbf{aab} \\
y = x \\
w = u \cdot \mathbf{b} \\
x \cdot v = v \cdot w \\
x \cdot v \neq w
\end{gathered}
$$


Length

Cong Closure

Normalize Eq

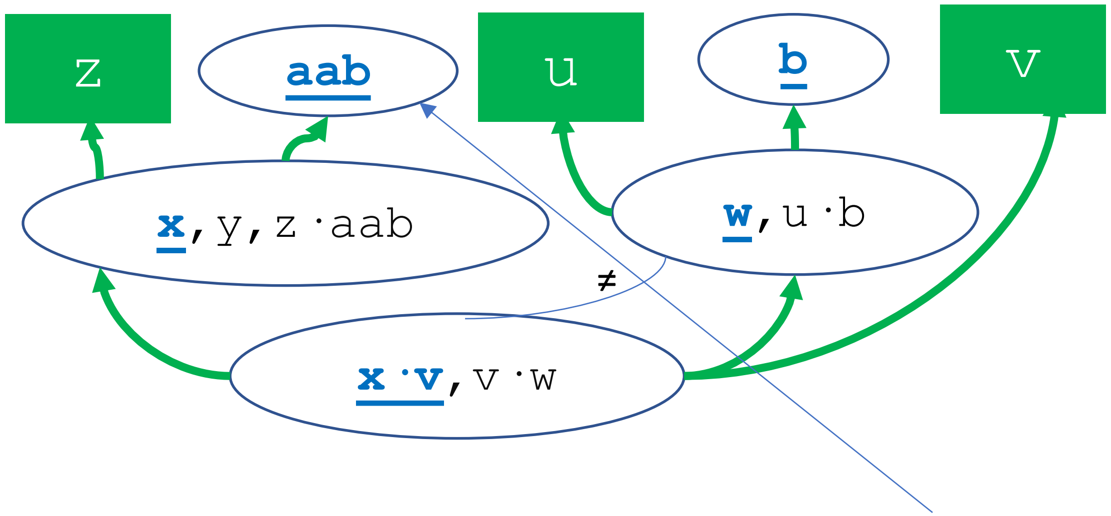
Normalize Deq

Cardinality

Normal forms computed bottom-up

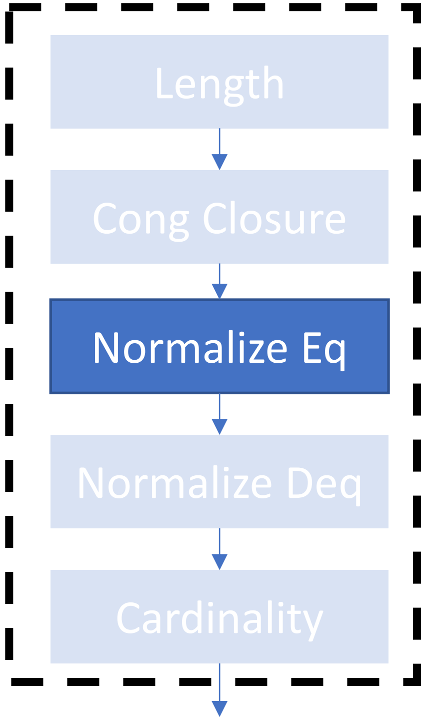
- First, compute containment relation induced by concatenation terms

  This relation is guaranteed to be acyclic due to length elaboration step (cycle ⇒ LIA-conflict)
- Base case: eq classes with just variables can be assigned representative as a normal form
- Inductive case: compare the *expanded* forms $t_1, \ldots, t_n$ of each non-variable
  - If $t_1 \cong \ldots \cong t_n$, assign one. If there exists distinct t_i, t_j, then try to equate them



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

Single non-variable string term ⇒ assign

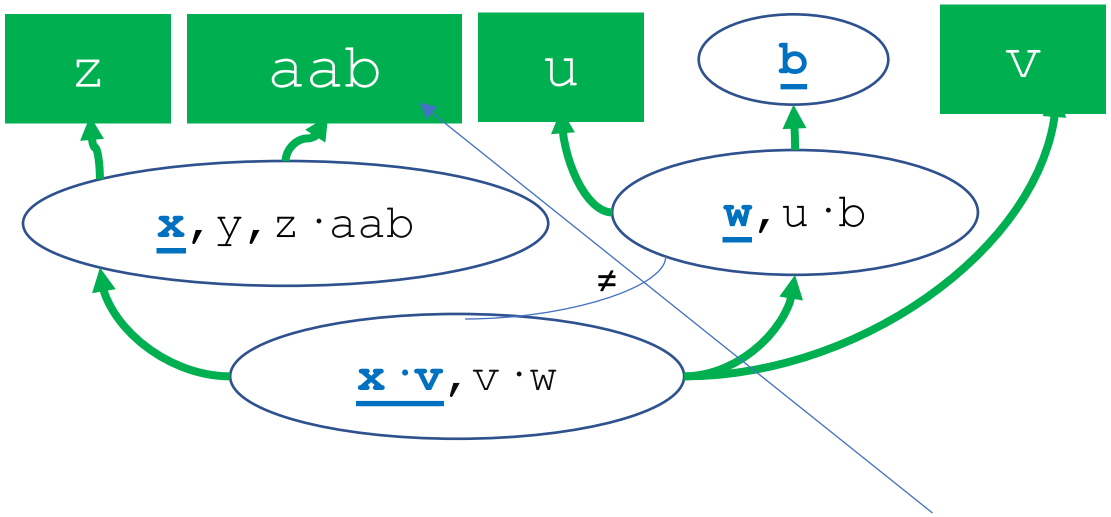

Length

Cong Closure

Normalize Eq

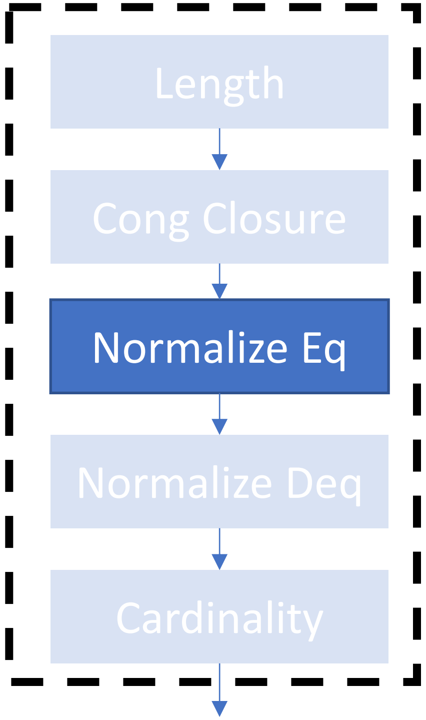
Normalize Deq

Cardinality



$$
\begin{gathered}
x = z \cdot \mathbf{aab} \\
y = x \\
w = u \cdot \mathbf{b} \\
x \cdot v = v \cdot w \\
x \cdot v \neq w
\end{gathered}
$$

Single non-variable string term ⇒ assign

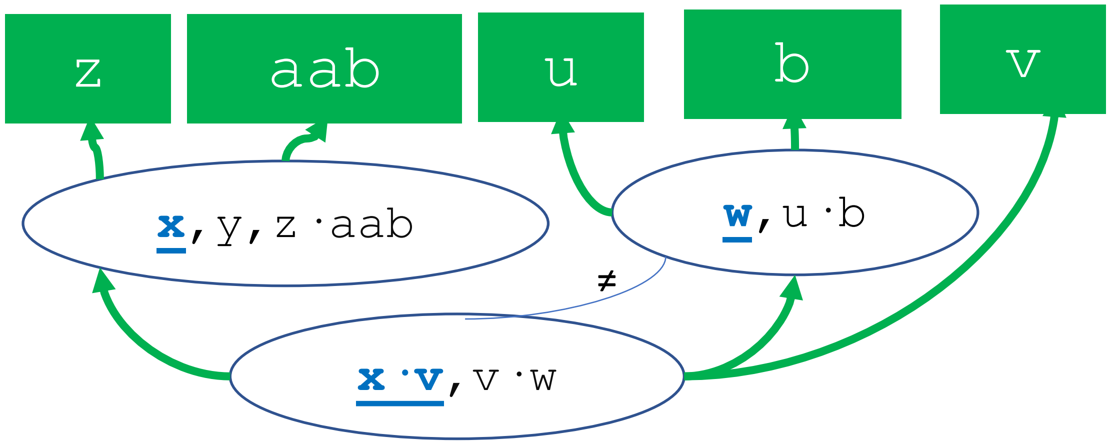

Length

Cong Closure

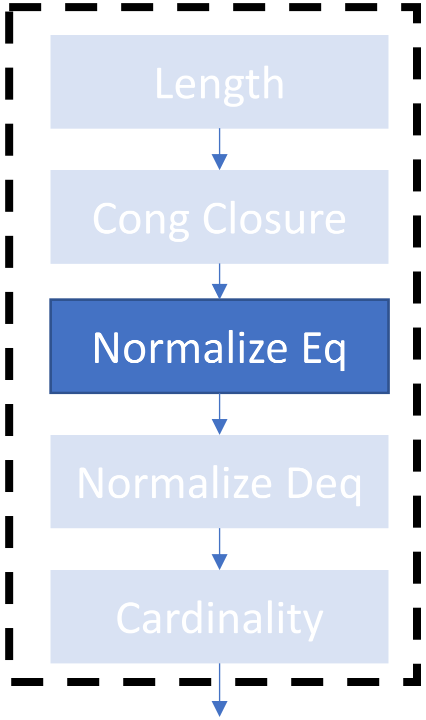
Normalize Eq

Normalize Deq

Cardinality



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

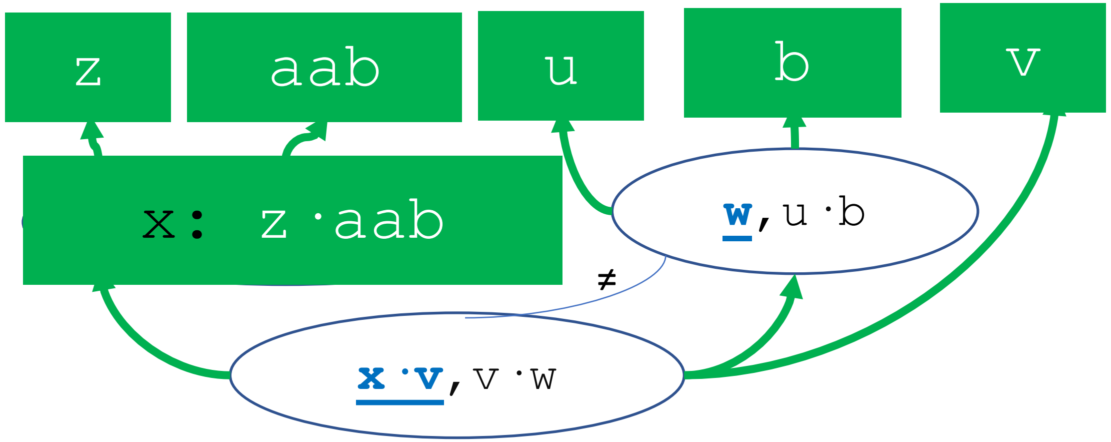

Length

Cong Closure

Normalize Eq

Normalize Deq

Cardinality



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

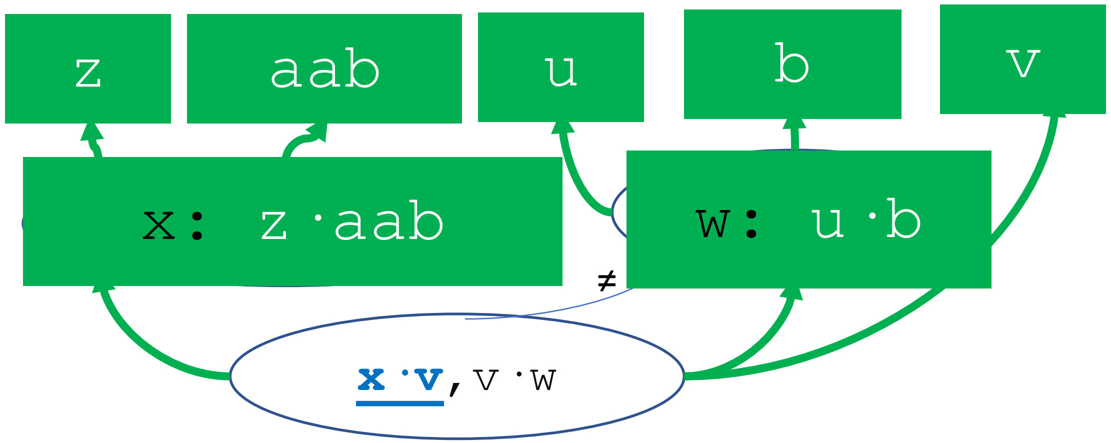
Length

Cong Closure

Normalize Eq

Normalize Deq

Cardinality



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

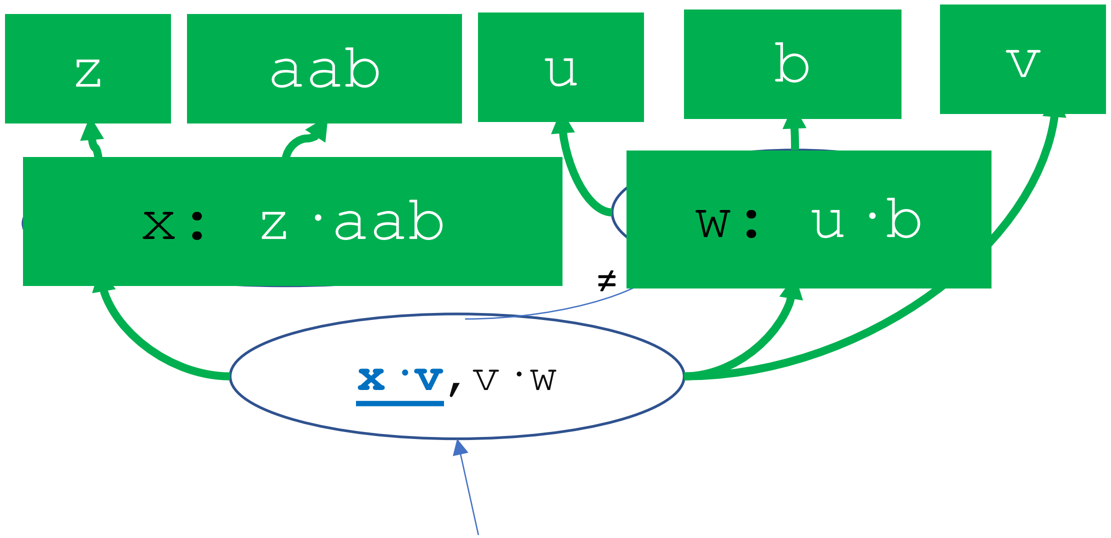
Length

Cong Closure

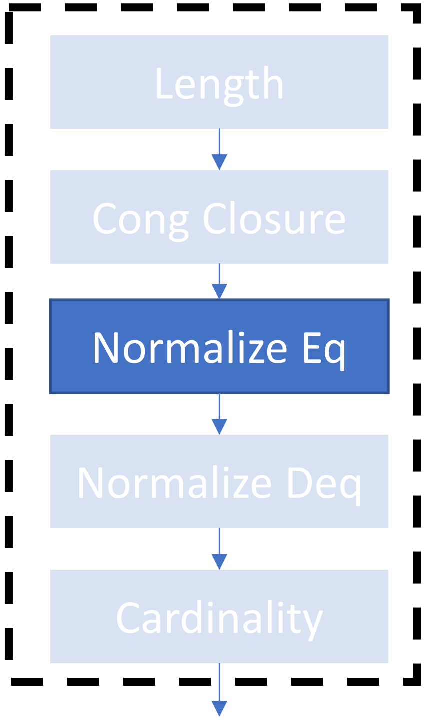
Normalize Eq

Normalize Deq

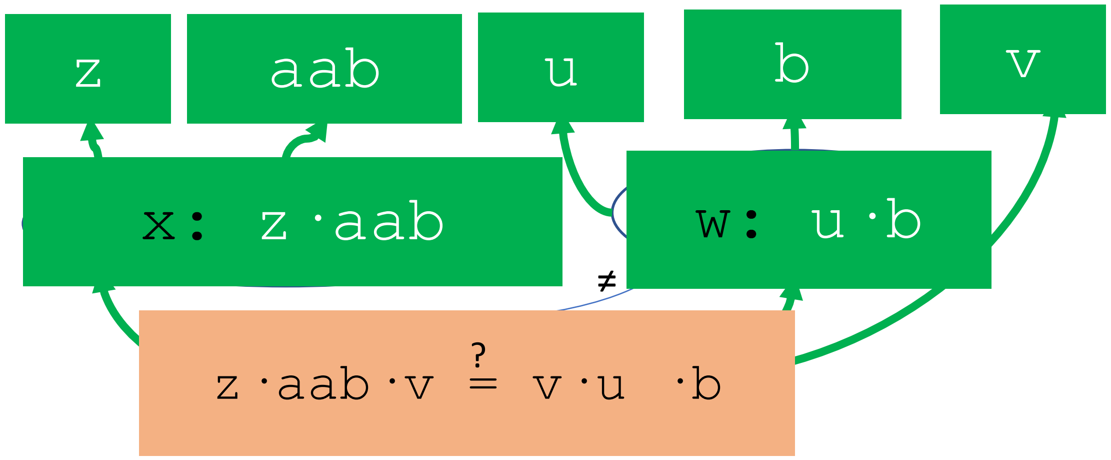
Cardinality



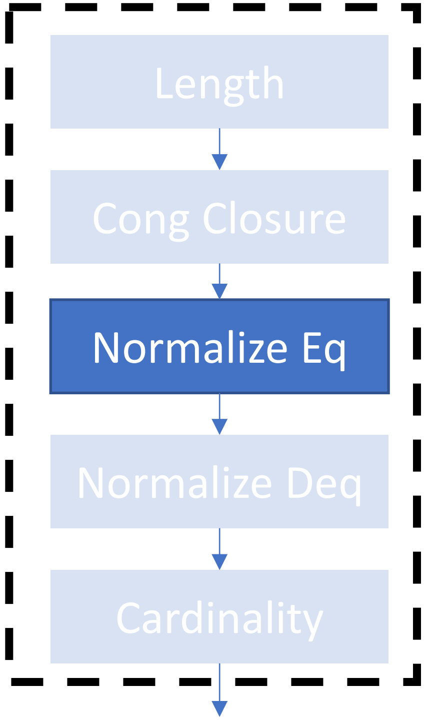
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$



- Equivalence class with two non-variable terms with distinct expanded forms:
  - $x \cdot v \ = \ (z \cdot \mathbf{aab}) \cdot v \ \ = \ z \cdot \mathbf{aab} \cdot v$
  - $v \cdot w \ = \ v \cdot (u \cdot \mathbf{b}) \ \ = \ v \cdot u \cdot \mathbf{b}$



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

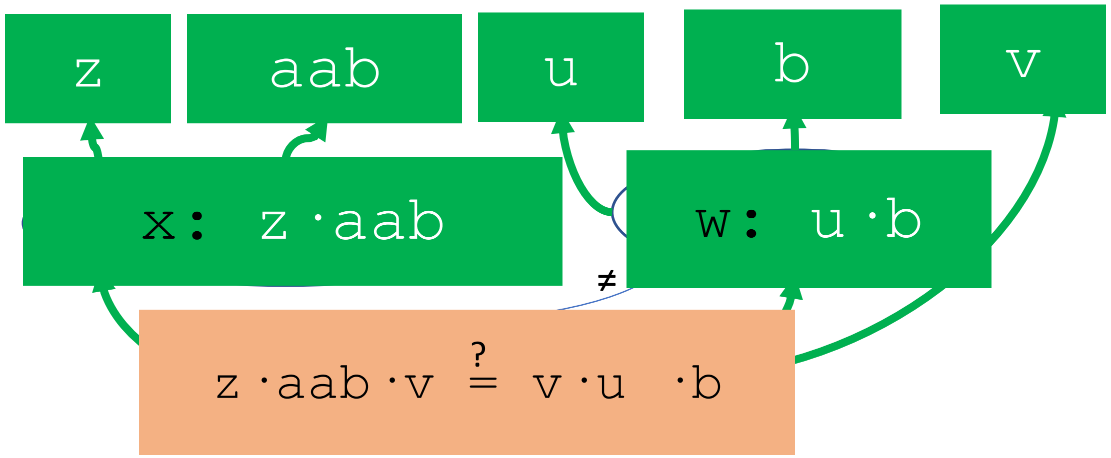

Length

Cong Closure

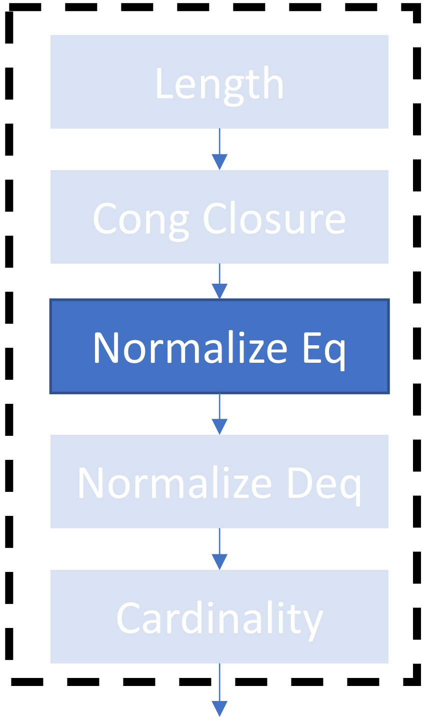
Normalize Eq

Normalize Deq

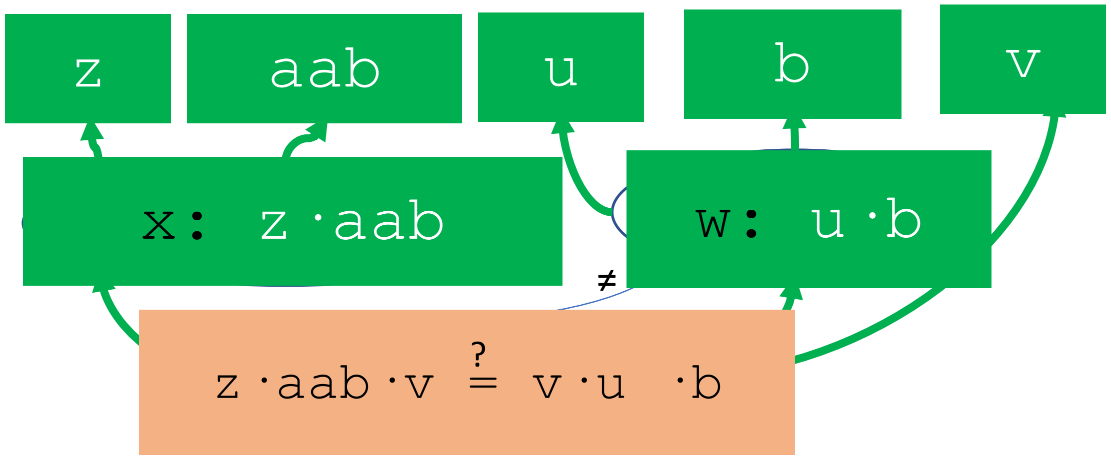
Cardinality



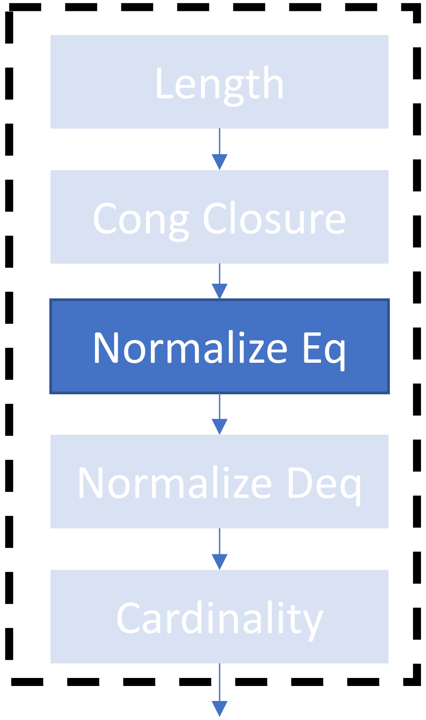
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$



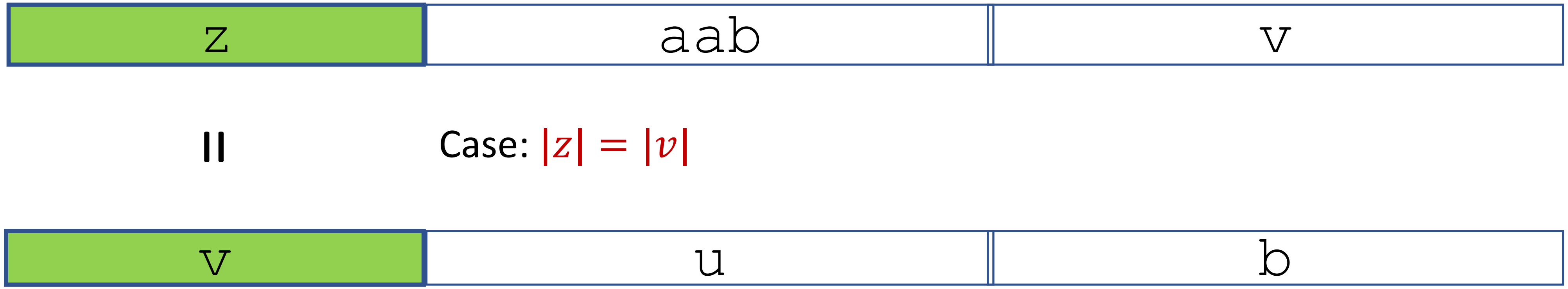
Goal: split strings so that all aligning components are equal



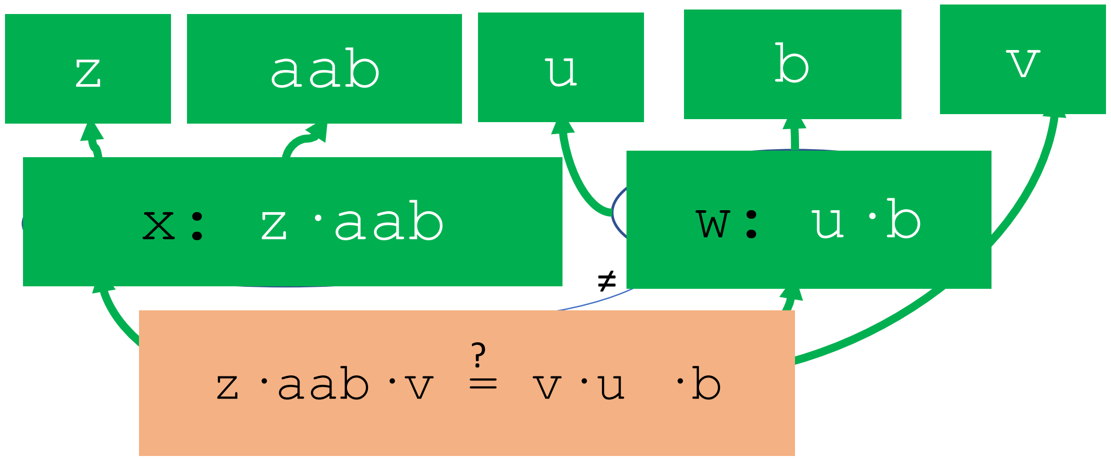
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$



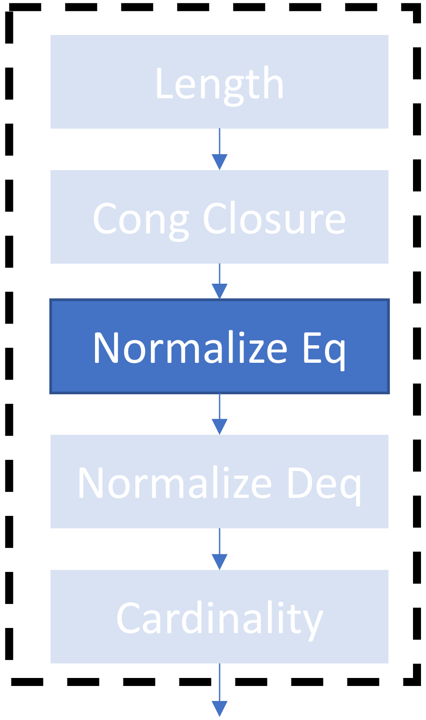
Consider three cases for making these two terms equal:



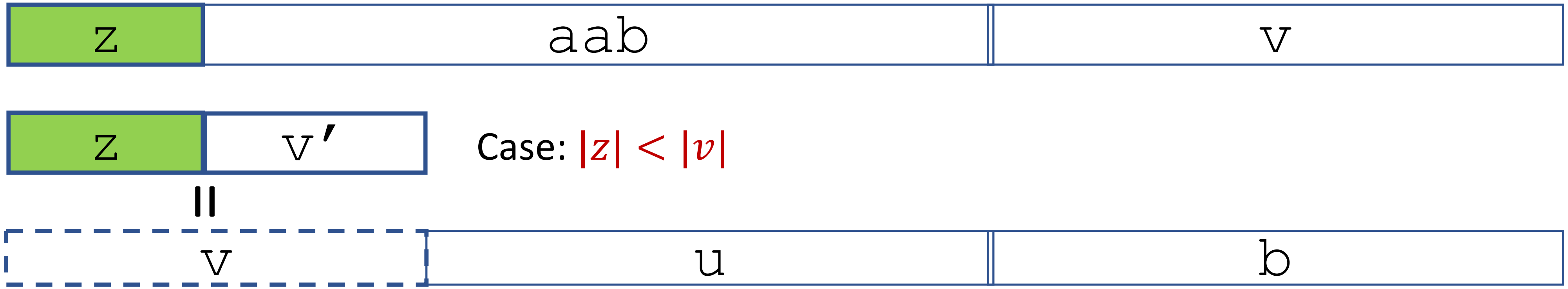
Case: $|z| = |v|$



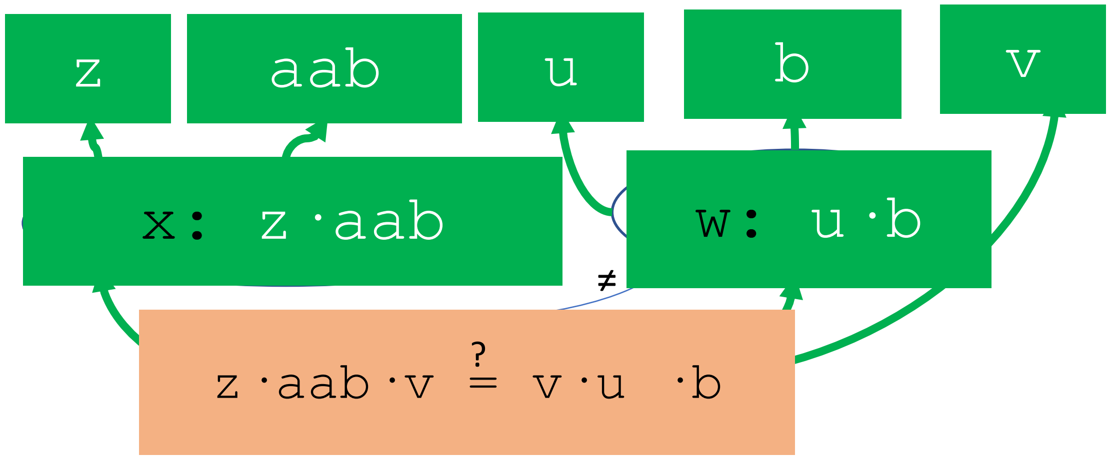
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$



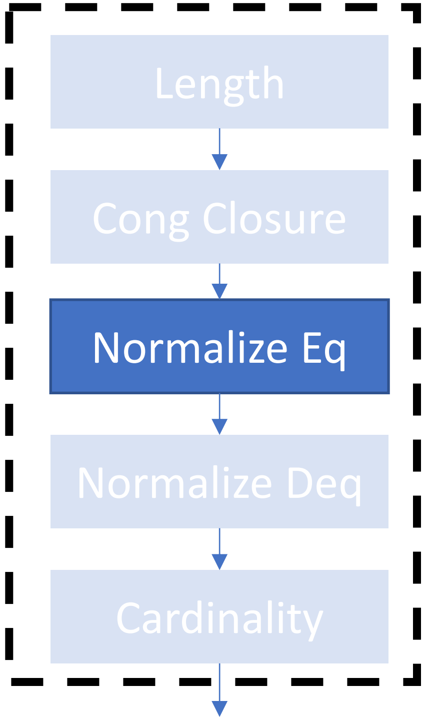
Consider three cases for making these two terms equal:



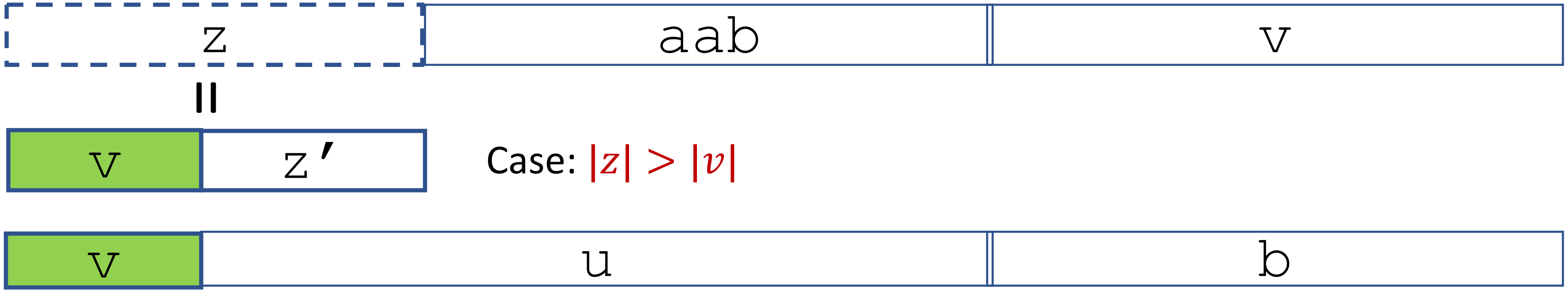
Case: $|z| < |v|$



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$

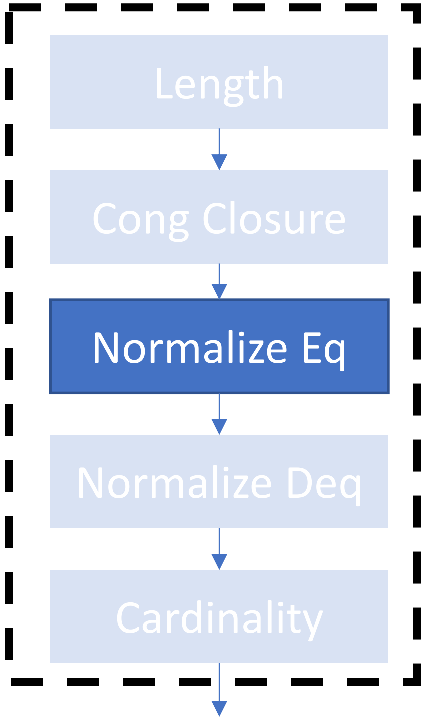


Consider three cases for making these two terms equal:



Case: $|z| > |v|$

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$
$$z = v$$


Length

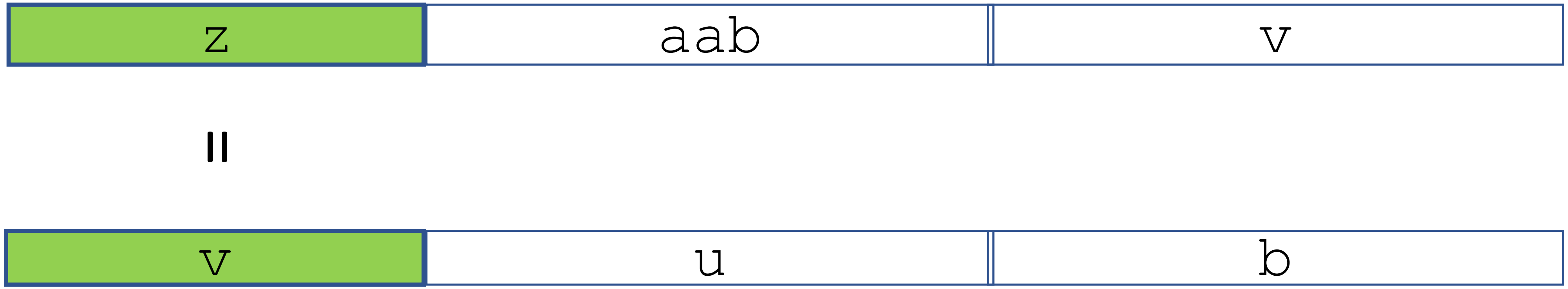
Cong Closure

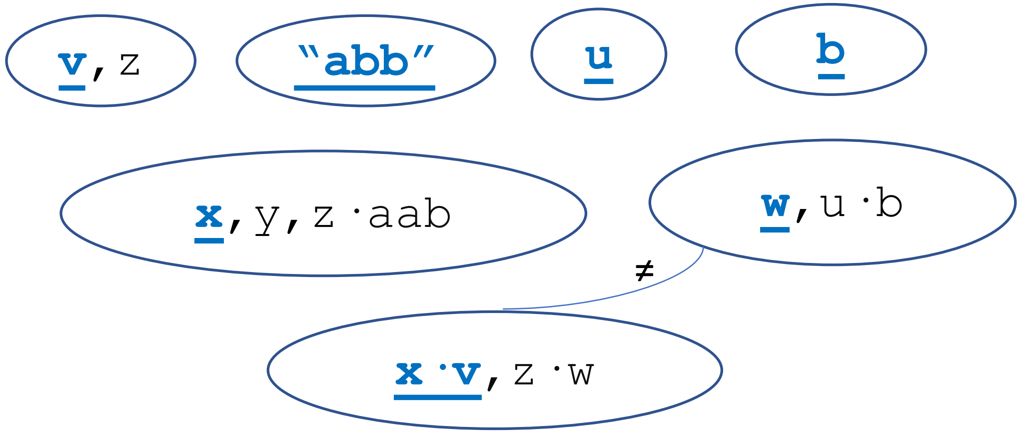
Normalize Eq

Normalize Deq

Cardinality

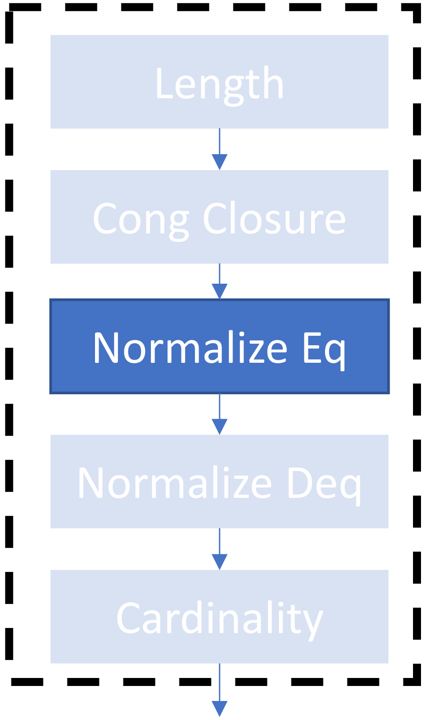
Equal case:



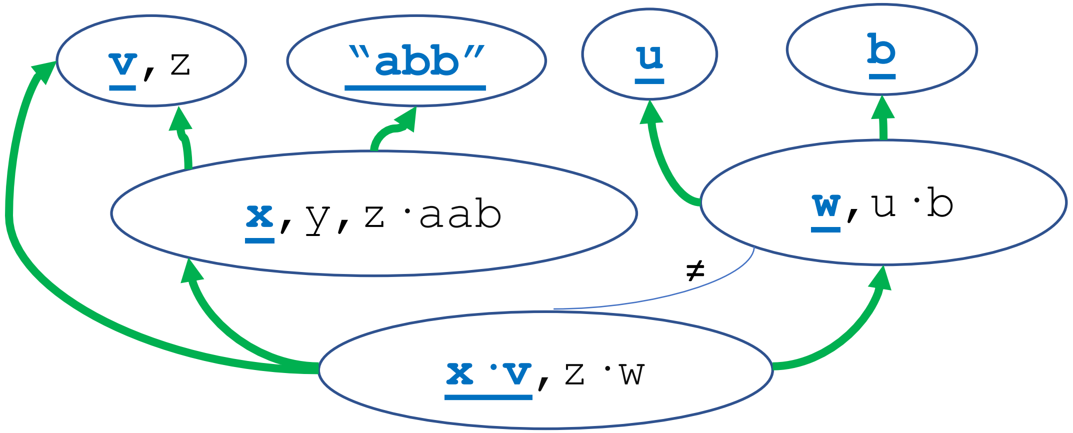


Equal case:

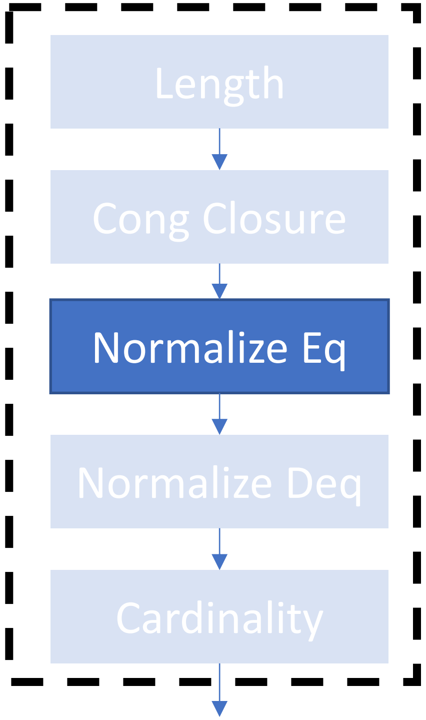
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$
$$z = v$$



Recompute congruence closure



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$
$$z = v$$


Length

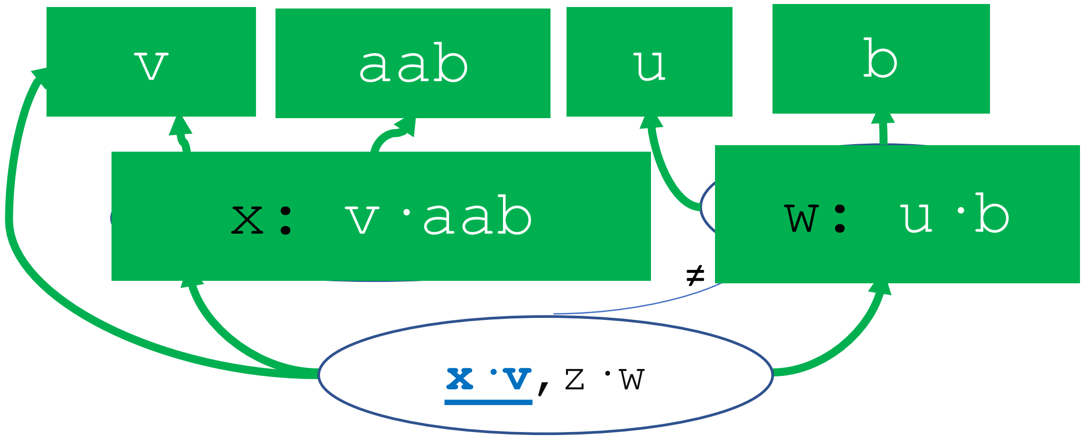
Cong Closure

Normalize Eq

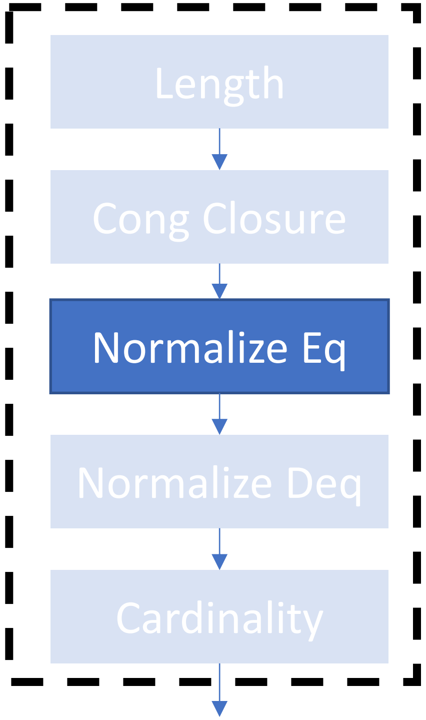
Normalize Deq

Cardinality

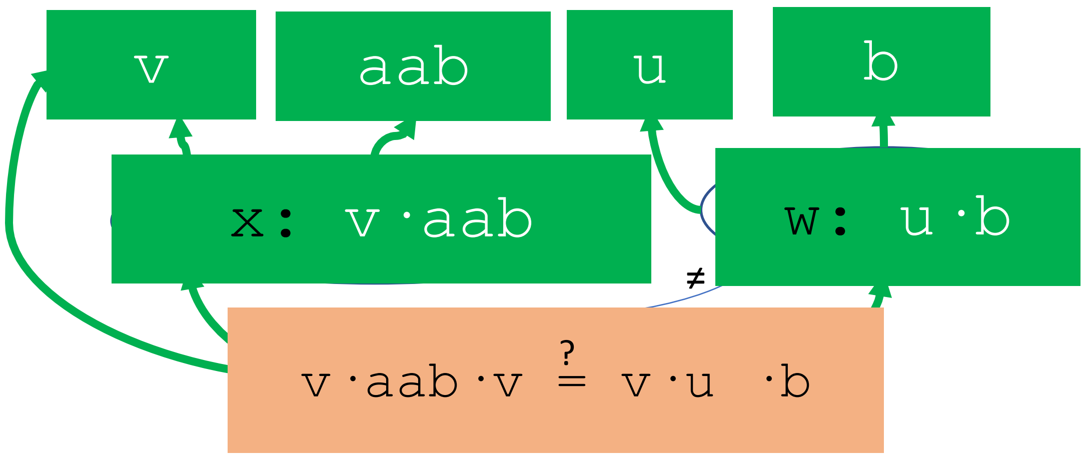
Recompute congruence closure and normal forms



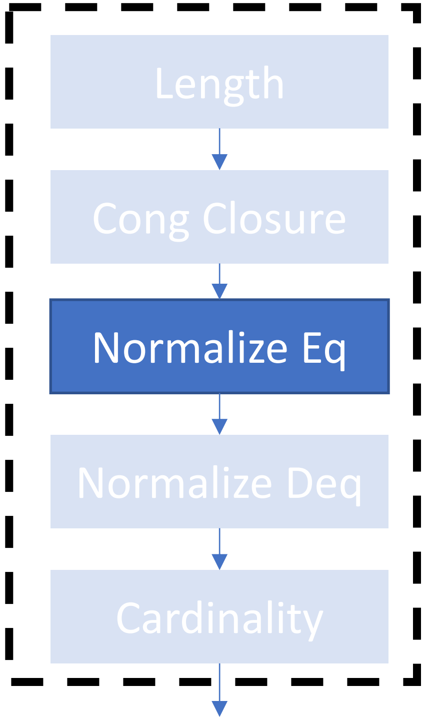
$$
\begin{gathered}
x = z \cdot \mathbf{aab} \\
y = x \\
w = u \cdot \mathbf{b} \\
x \cdot v = v \cdot w \\
x \cdot v \neq w \\
z = v
\end{gathered}
$$

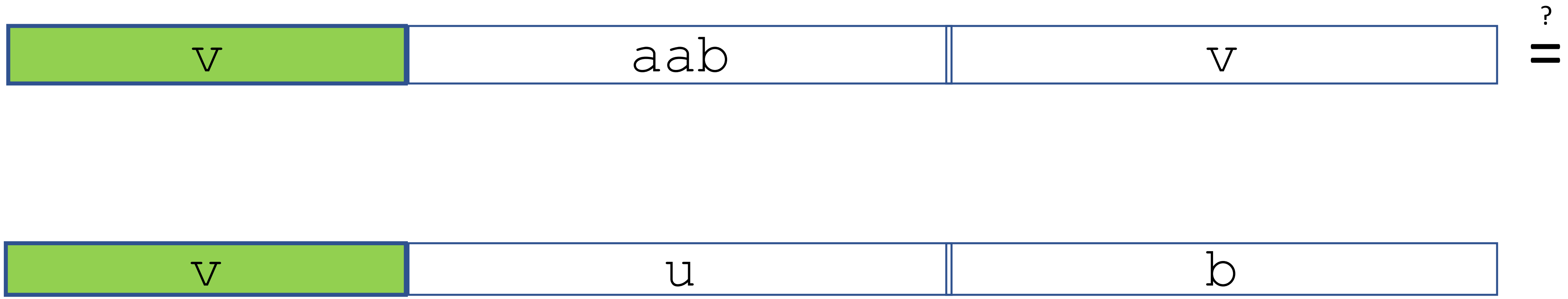


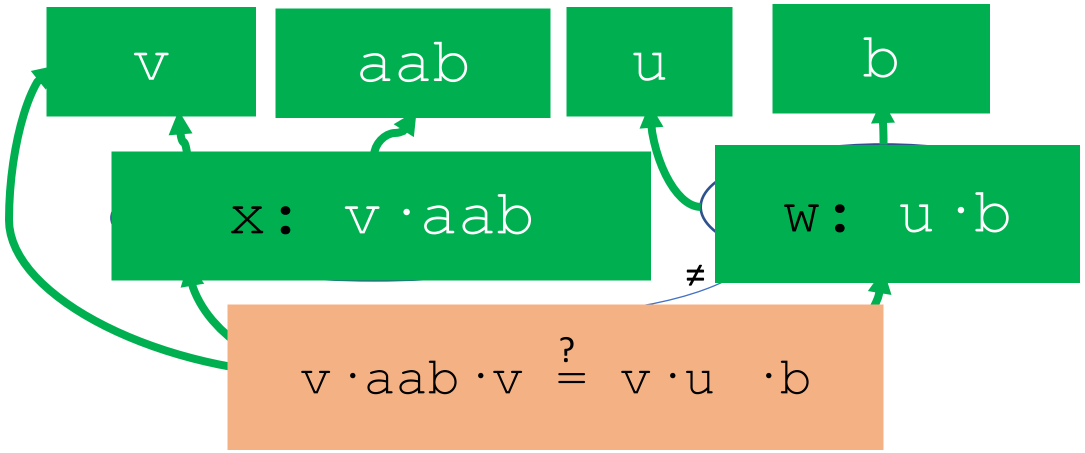
Recompute congruence closure and normal forms



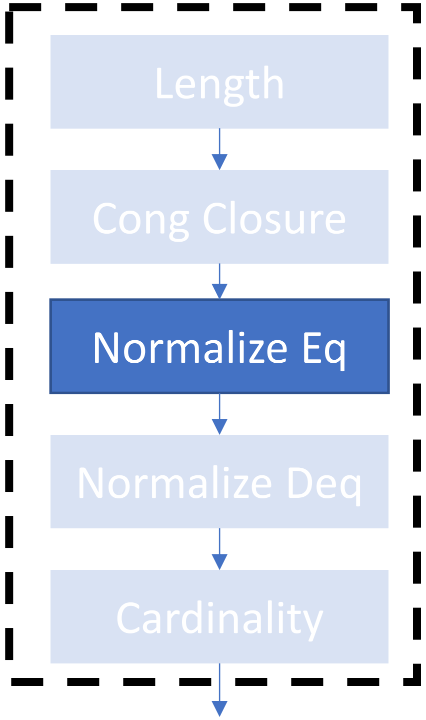
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v = v \cdot w$$
$$x \cdot v \neq w$$
$$z = v$$

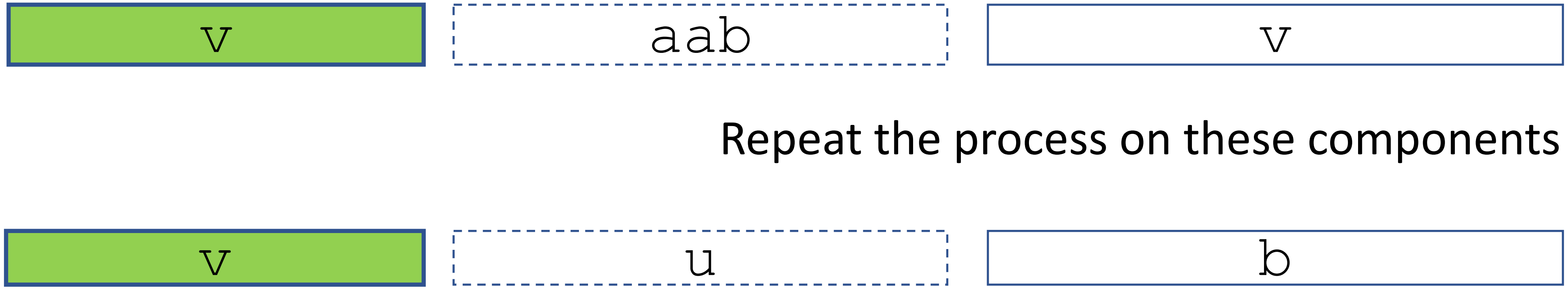






$$
\begin{aligned}
x &= z \cdot \mathbf{aab} \\
y &= x \\
w &= u \cdot \mathbf{b} \\
x \cdot v &= v \cdot w \\
x \cdot v &\neq w \\
z &= v
\end{aligned}
$$


Length
Cong Closure
Normalize Eq
Normalize Deq
Cardinality



Repeat the process on these components

Splitting on String Equalities

Choice of equalities is quite sophisticated and critical to performance:

1. Prefers propagations over splits
   E.g., $x \cdot w = y \cdot w \;\Rightarrow\; x = y$ over $x \cdot w = z \cdot v \;\Rightarrow\; (x = z \cdot x' \vee z = x \cdot z')$
2. Considers both the prefix and suffix of strings
   E.g., $w \cdot x = w \cdot y \;\Rightarrow\; x = y$
3. Exploits length entailment [Zheng et al., 2015]
   If $|x| > |y|$ according to the arithmetic solver,
   then $x \cdot w = y \cdot v \wedge |x| > |y| \;\Rightarrow\; x = y \cdot x'$

Splitting on String Equalities

Choice of equalities is quite sophisticated and critical to performance:

4. Propagates constraints based on adjacent constants

   E.g., $x \cdot \mathbf{b} = \mathbf{aab} \cdot y \;\Rightarrow\; x = \mathbf{aa} \cdot x'$, since $\mathbf{b}$ cannot overlap with prefix $\mathbf{aa}$

5. Treats looping word equations specially [Liang et al., 2014]

   Splitting leads to non-termination; instead, reduce to RE membership

   E.g., $x \cdot \mathbf{ba} = \mathbf{ab} \cdot x \;\Rightarrow\; x \in (\mathbf{ab})^{*}\mathbf{a}$

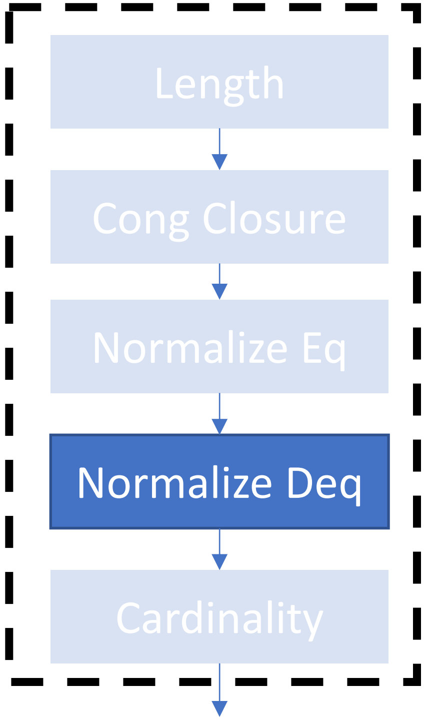
String Solver: Normalize Disequalities

$$x = z \cdot \mathbf{aab}$$

$$y = x$$

$$w = u \cdot \mathbf{b}$$

$$x \cdot v \neq v \cdot w$$

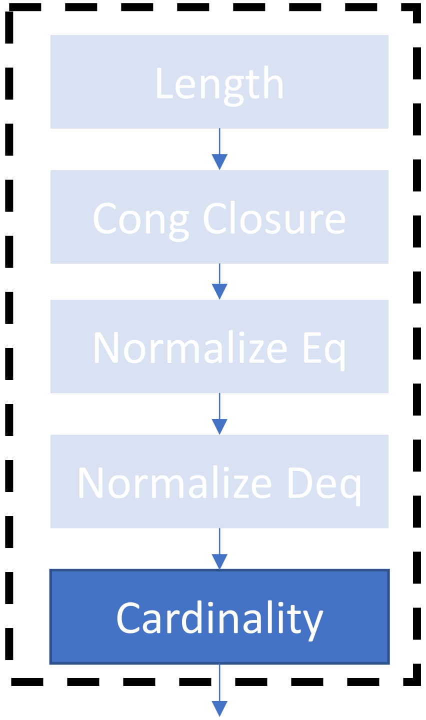

Length
Cong Closure
Normalize Eq
Normalize Deq
Cardinality

Disequalities are handled analogously to equalities

- If $|x \cdot v| \neq |v \cdot w|$, then trivially $x \cdot v \neq v \cdot w$
- Otherwise, consider the normal forms of $x \cdot v$ and $v \cdot w$ from previous step
- **Goal:** find any two aligning components that are disequal

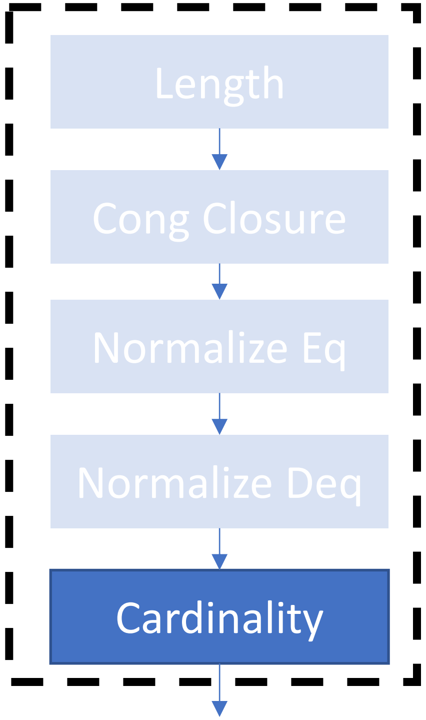
5. Check Cardinality Constraints

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$



5. Check Cardinality Constraints

$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$



$\mathcal{M}_S$ may be unsatisfiable because Σ is finite

Example:

- Σ consists of 256 characters, and
- $\mathcal{M}_S$ entails that 257 distinct strings of length 1 exist

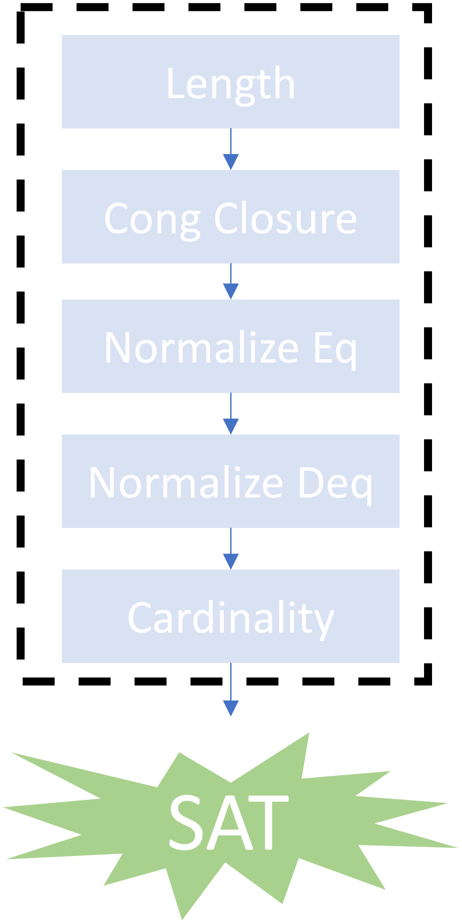
$$\text{distinct}(s_1, \dots, s_{257}), |s_1| = 1, \dots, |s_{257}| = 1 \models \bot$$

Finally: Compute Model

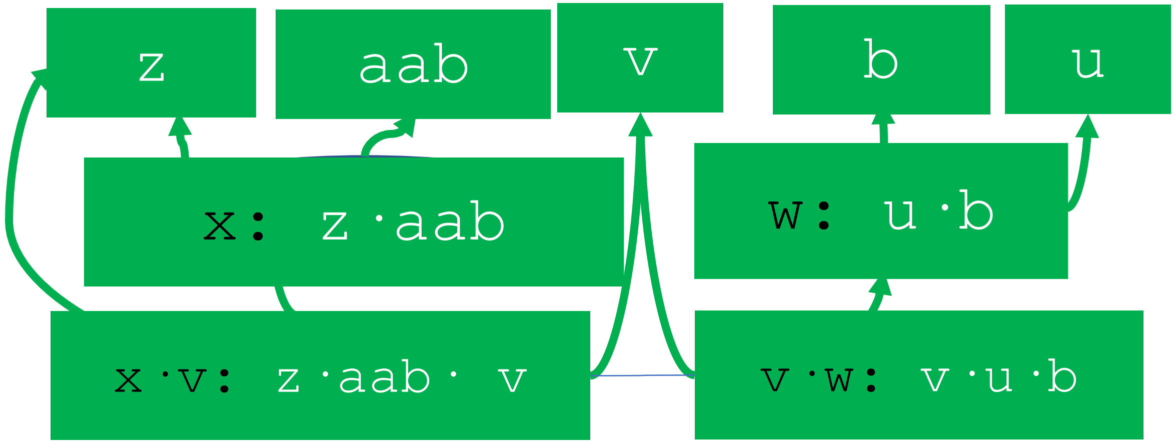
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$

If all steps finish with
no new lemmas:

- $\mathcal{M}_s$ is T_s-satisfiable
- Compute model based on normal forms
  - assign string constants to eq classes whose normal form is a variable
    - Length fixed by model from arithmetic solver
  - Interpret each var as the value of its eq class' normal form


Length
Cong Closure
Normalize Eq
Normalize Deq
Cardinality
SAT

Compute Model



z

aab

v

b

u

x: z·aab

w: u·b

x·v: z·aab· v

v·w: v·u·b

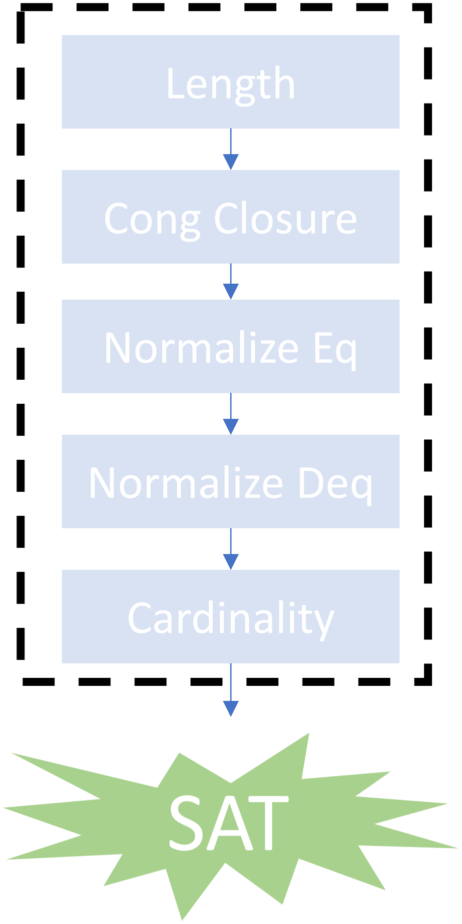
$$x = z \cdot \mathbf{aab}$$

$$y = x$$

$$w = u \cdot \mathbf{b}$$

$$x \cdot v \neq v \cdot w$$

$$v \neq z$$



Length

Cong Closure

Normalize Eq

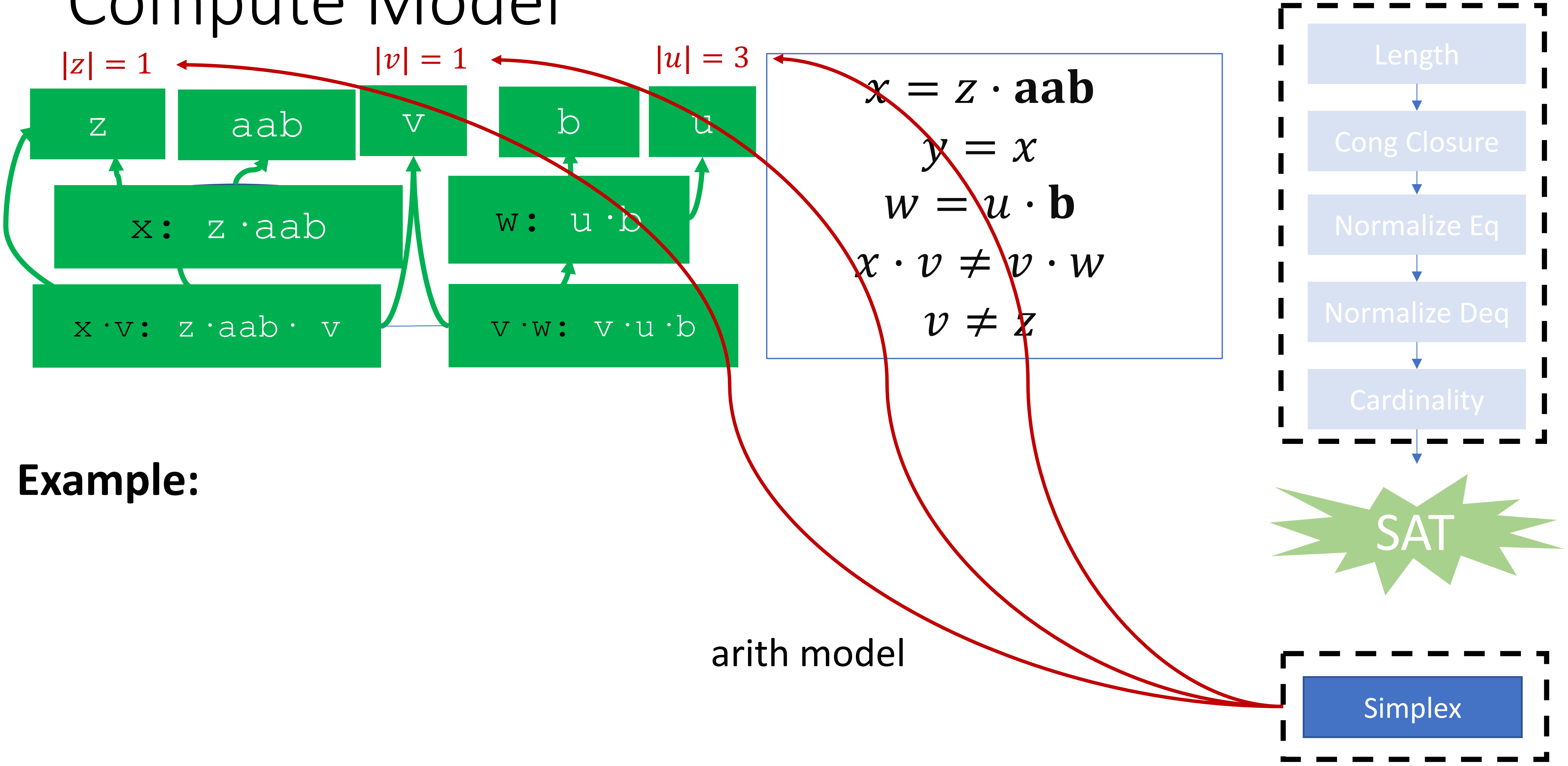
Normalize Deq

Cardinality

SAT

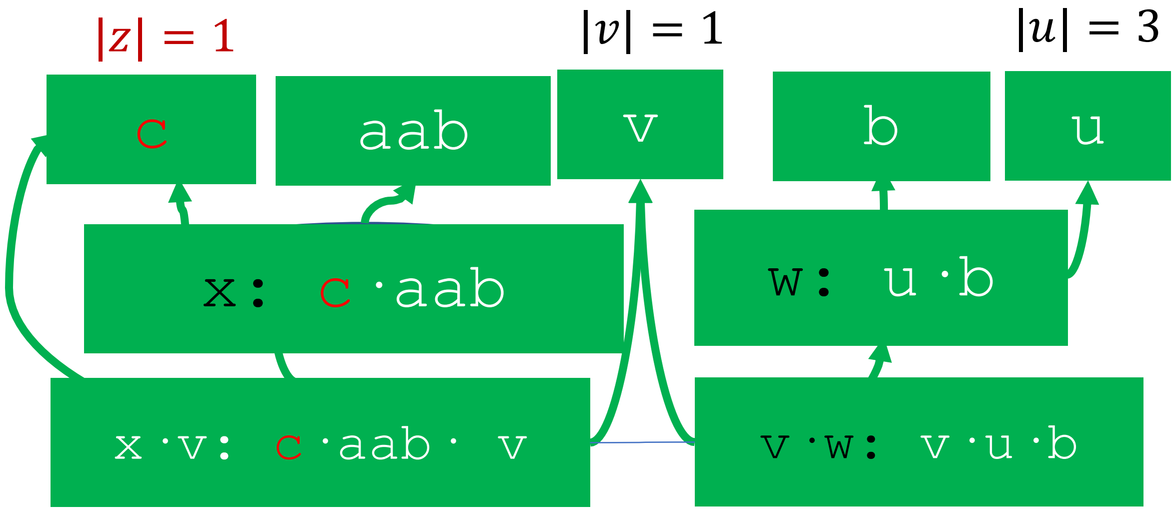
Compute Model

Example:

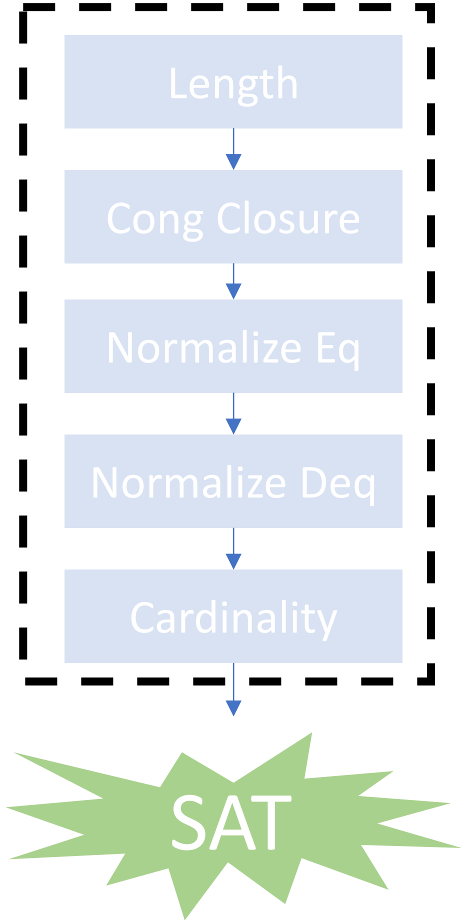


$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$

Compute Model



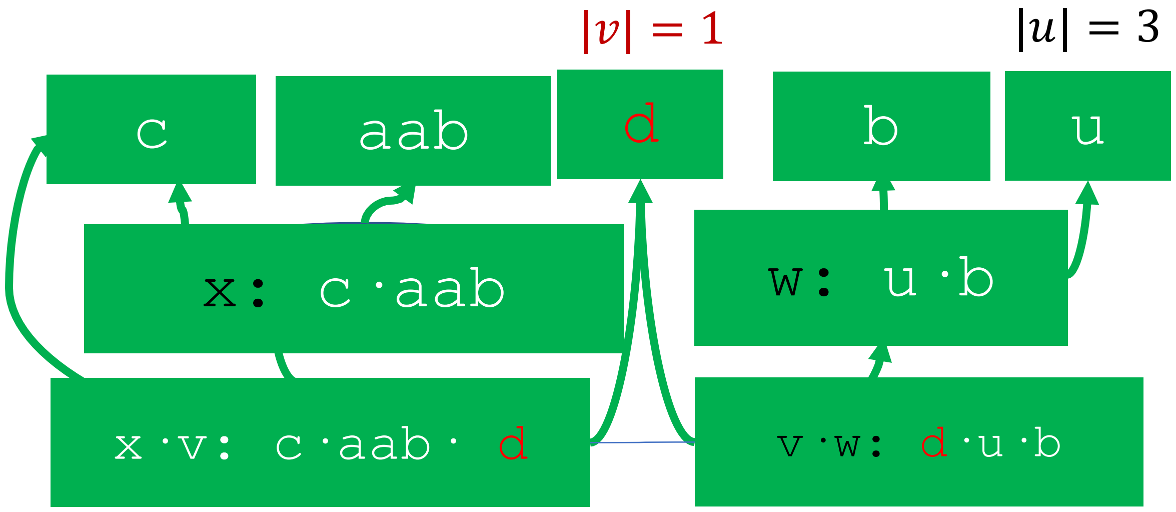
$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$



Example:

- $z \longmapsto \mathbf{c}$

Compute Model



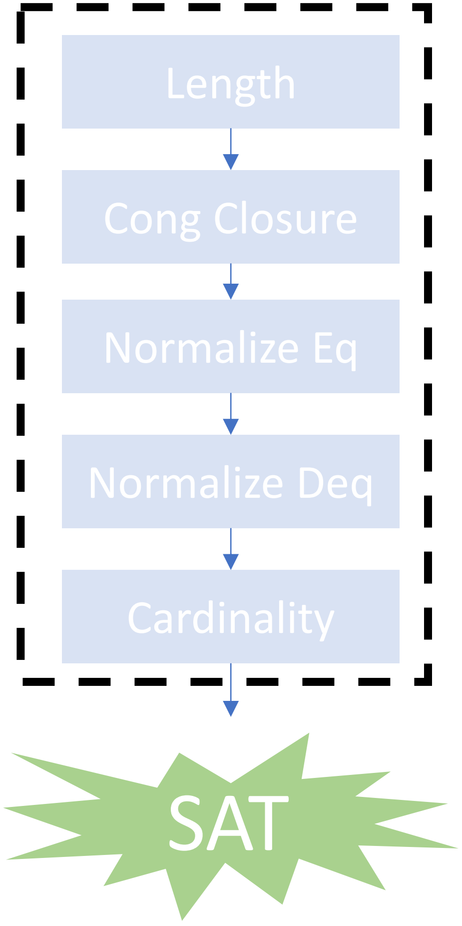
$$x = z \cdot \mathbf{aab}$$

$$y = x$$

$$w = u \cdot \mathbf{b}$$

$$x \cdot v \neq v \cdot w$$

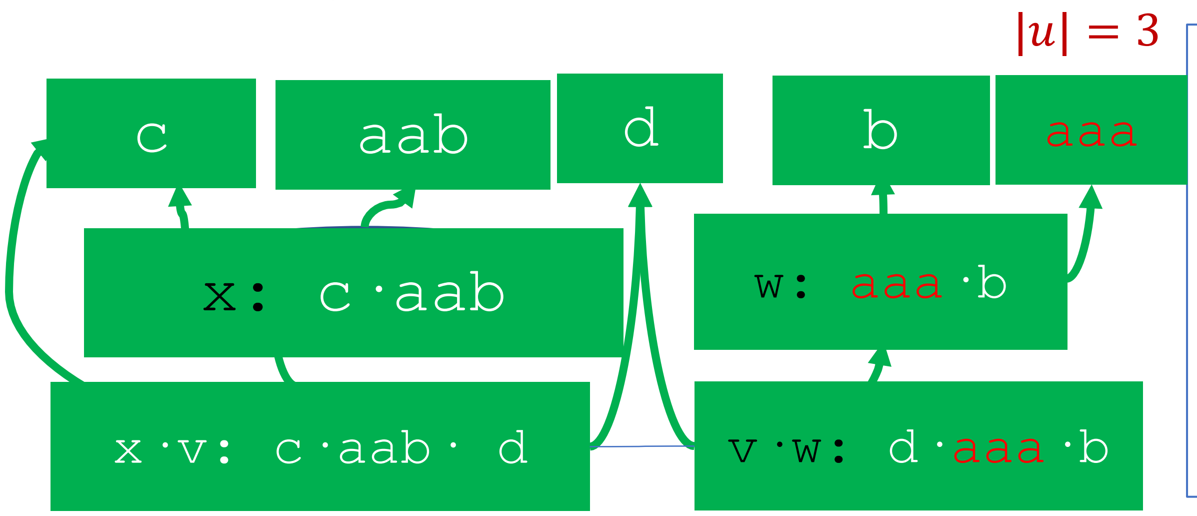
$$v \neq z$$



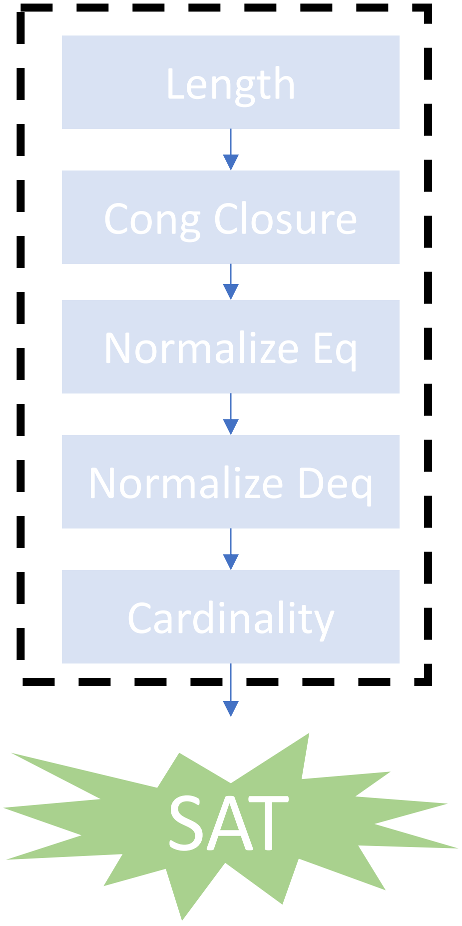
Example:

- $z \longmapsto \mathbf{c}$
- $v \longmapsto \mathbf{d}$

Compute Model



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$

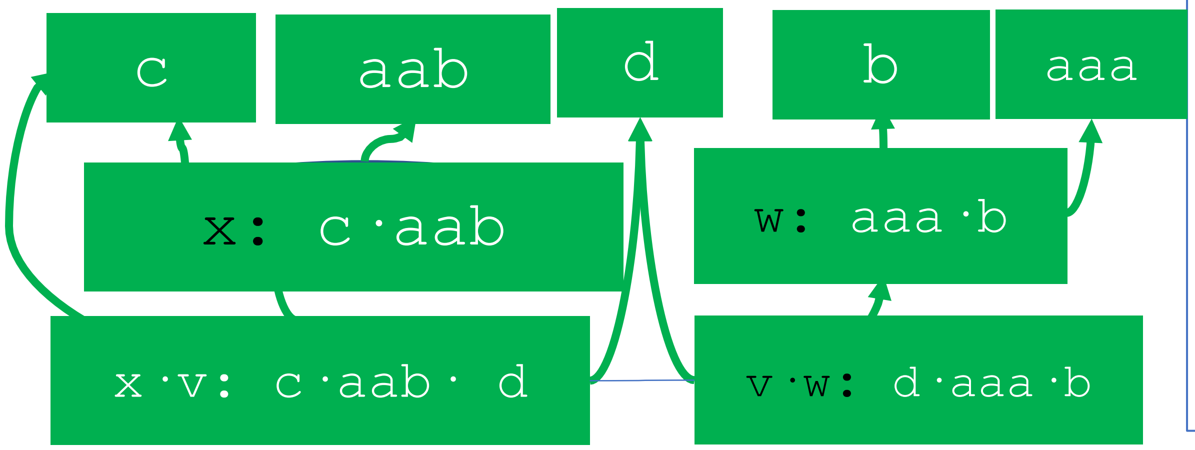


Example:

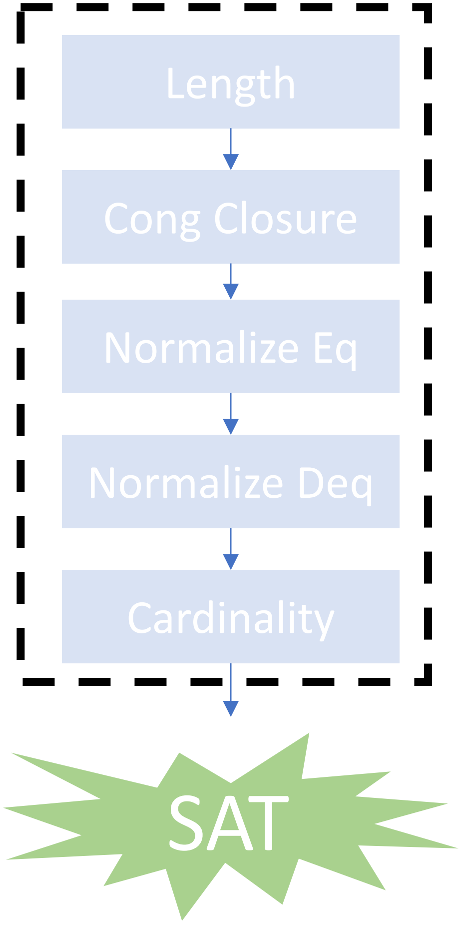
- $z \longmapsto \mathbf{c}$
- $v \longmapsto \mathbf{d}$
- $u \longmapsto \mathbf{aaa}$

Check-cardinality step ensures there are enough constants

Compute Model



$$x = z \cdot \mathbf{aab}$$
$$y = x$$
$$w = u \cdot \mathbf{b}$$
$$x \cdot v \neq v \cdot w$$
$$v \neq z$$



Example:

- $z \longmapsto \mathbf{c}$
- $v \longmapsto \mathbf{d}$
- $u \longmapsto \mathbf{aaa}$
- Other vars assigned to value of the normal form of their eq classes

$$x \longmapsto \mathbf{caab} \quad y \longmapsto \mathbf{caab} \quad w \longmapsto \mathbf{aaab}$$

Saturation criterion for procedure ensures this model satisfies $\mathcal{M}_s$

Techniques for Fast String Solving in cvc5

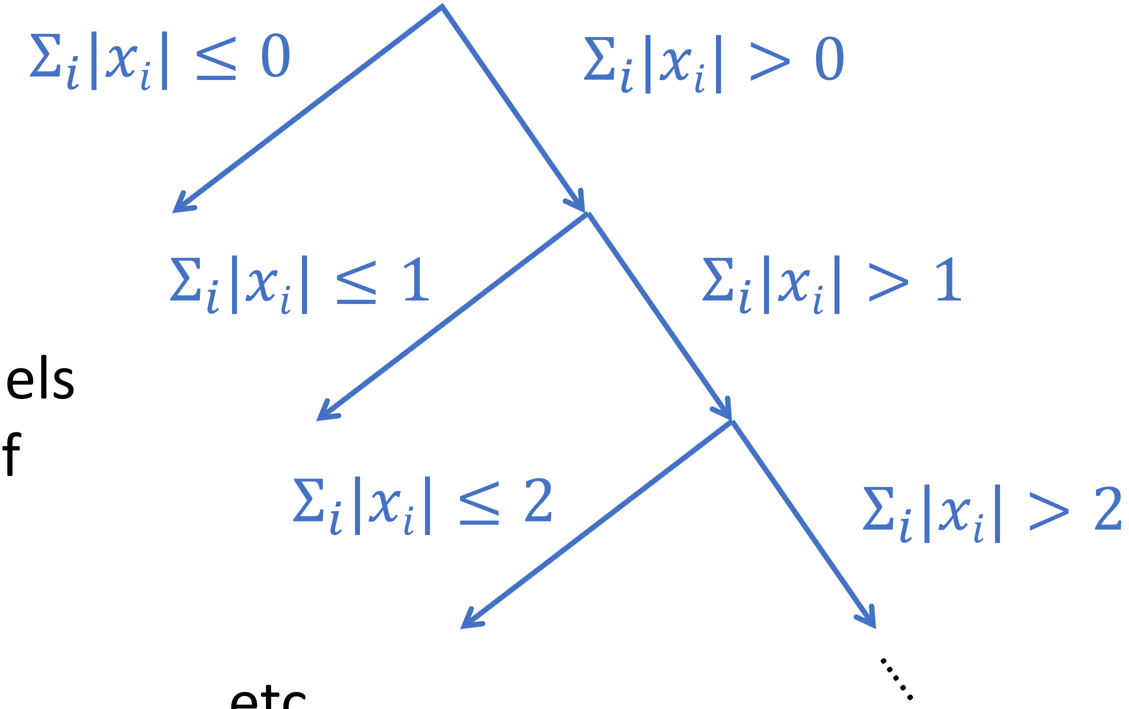
- Finite model finding
- Context-dependent simplification for extended constraints
- Witness sharing
- Regular expression elimination
- String to code point conversion

Finite Model Finding for Strings

Finite Model Finding for Strings

Idea: Incrementally bound the lengths of input string variables $x_1, \ldots, x_n$

$\Rightarrow$ Improves solver's solving time for problems with small models



Search for models
where sum of
lengths is 0

Search for models
where sum of
lengths is 1

etc.

Context-Dependent Simplification for Extended String Constraints

[Reynolds, Woo, Barrett, Brumley, Liang and Tinelli, CAV’17]

Extended String Constraint Language

Substring: $\mathrm{substr}(x, n, l)$

- the substring of string x starting at position n of length at most l

String contains: $\mathrm{contains}(x, y)$

- true iff string x contains y as a substring

Find index: $\mathrm{indexof}(x, n, p)$

- the position of the first occurrence of string y in x, starting from position n if any; -1 otherwise

String replace: $\mathrm{replace}(x, y, y')$

- the result of replacing the first occurrence of string y in x (if any) with y'

Example: $\neg\mathrm{contains}(\mathrm{substr}(x, 0{,}3), \mathbf{a}) \;\wedge\; 0 \leq \mathrm{indexof}(x, \mathbf{ab}, 0) < 4$

How do we handle Extended String Constraints?

$\neg\mathrm{contains}(x, \mathbf{a})$

How do we handle Extended String Constraints?

Naively, by reduction to basic constraints + bounded ∀

$\neg\mathrm{contains}(x, \mathbf{a})$

How do we handle Extended String Constraints?

Naively, by reduction to basic constraints + bounded ∀

$$\neg\text{contains}(x, \mathbf{a})$$

$$\downarrow$$

$$\forall\ 0 \leq n < |x|.\ \text{substr}(x, n, 1) \neq \mathbf{a}$$

Expand contains

How do we handle Extended String Constraints?

Naively, by reduction to basic constraints + bounded $\forall$

$$\neg\text{contains}(x, \mathbf{a})$$

$$\forall\ 0 \leq n < |x|.\ \text{substr}(x, n, 1) \neq \mathbf{a}$$

Expand contains

$$\text{substr}(x, 0,1) \neq \mathbf{a}\ \wedge\ \ldots\ \wedge\ \text{substr}(x, 4,1) \neq \mathbf{a}$$

Assuming bound $|x| \leq 5$

How do we handle Extended String Constraints?

Naively, by reduction to basic constraints + bounded ∀

$$\neg\text{contains}(x, \mathbf{a})$$

$$\forall\ 0 \leq n < |x|.\ \text{substr}(x, n, 1) \neq \mathbf{a}$$

Expand contains

$$\text{substr}(x, 0{,}1) \neq \mathbf{a}\ \wedge\ \ldots\ \wedge\ \text{substr}(x, 4{,}1) \neq \mathbf{a}$$

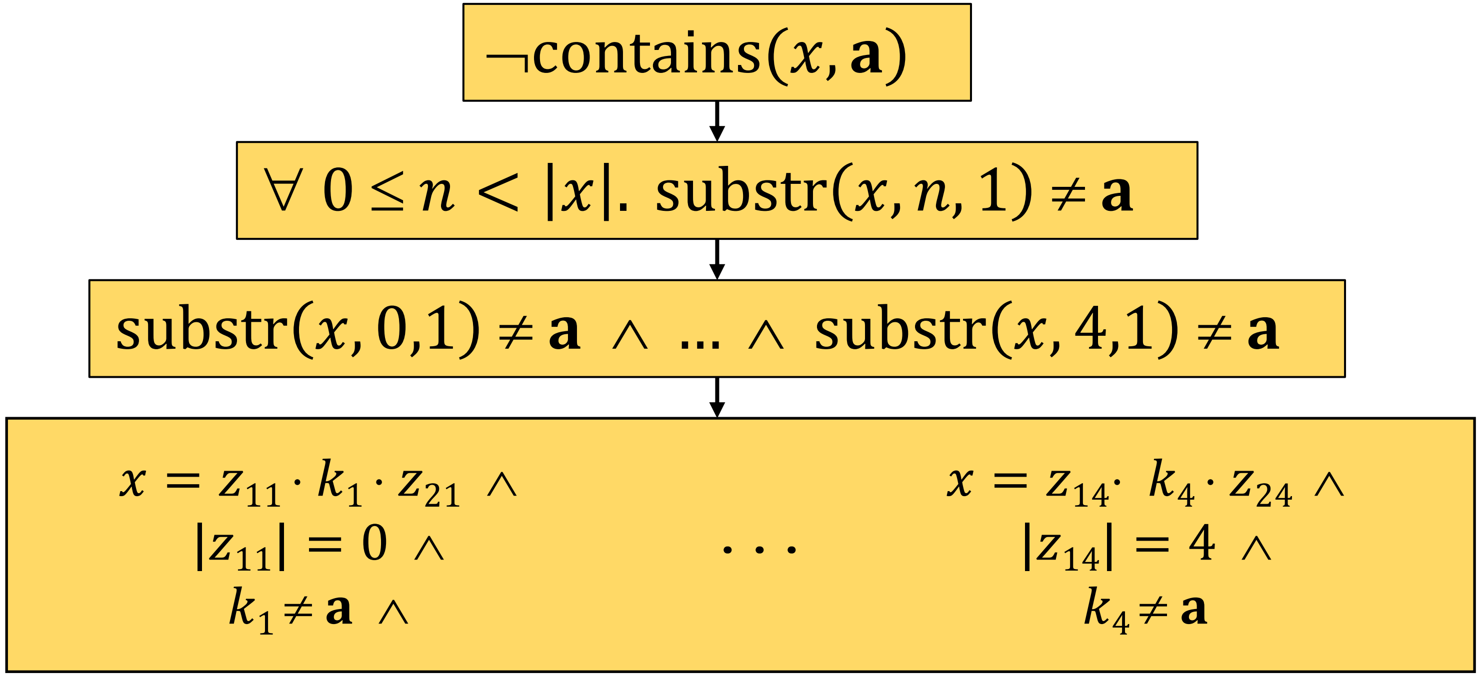
Assuming bound $|x| \leq 5$

$$\begin{array}{ccc} x = z_{11} \cdot k_1 \cdot z_{21}\ \wedge & & x = z_{14} \cdot\ k_4 \cdot z_{24}\ \wedge \\ |z_{11}| = 0\ \wedge & \ldots & |z_{14}| = 4\ \wedge \\ k_1 \neq \mathbf{a}\ \wedge & & k_4 \neq \mathbf{a} \end{array}$$

Expand substr

How do we handle Extended String Constraints?

Naively, by reduction to basic constraints + bounded $\forall$



$\neg\text{contains}(x, \mathbf{a})$

$\forall\ 0 \leq n < |x|.\ \text{substr}(x, n, 1) \neq \mathbf{a}$

Expand contains

$\text{substr}(x, 0{,}1) \neq \mathbf{a}\ \wedge\ \ldots\ \wedge\ \text{substr}(x, 4{,}1) \neq \mathbf{a}$

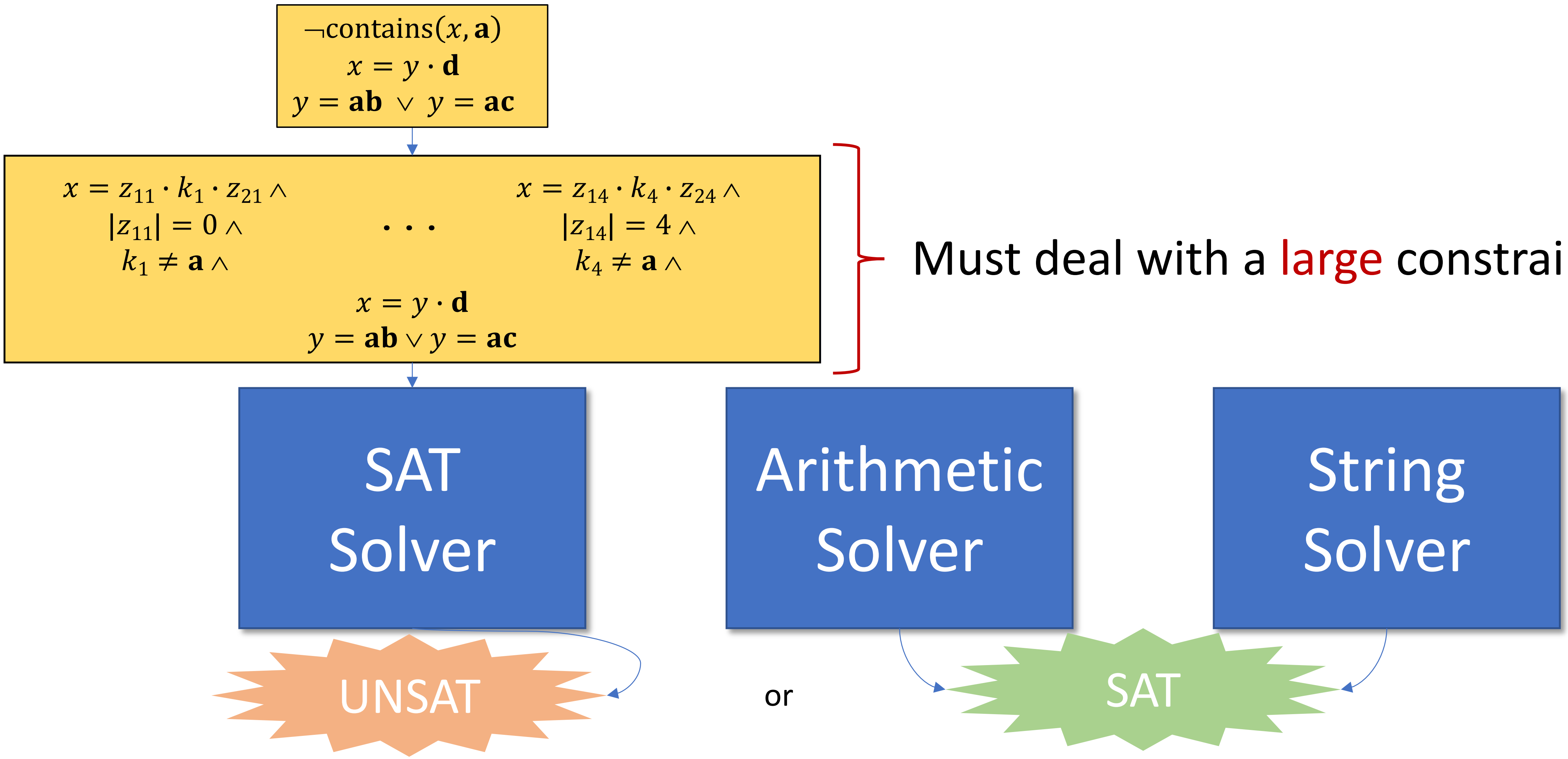
Assuming bound $|x| \leq 5$

$$x = z_{11} \cdot k_1 \cdot z_{21}\ \wedge\ |z_{11}| = 0\ \wedge\ k_1 \neq \mathbf{a}\ \wedge \quad \cdots \quad x = z_{14} \cdot k_4 \cdot z_{24}\ \wedge\ |z_{14}| = 4\ \wedge\ k_4 \neq \mathbf{a}$$

Expand substr

Approach followed by many solvers [Bjorner et al. 2009, Zheng et al. 2013, Li et al. 2013, Trinh et al. 2014]

(Eager) Expansion of Extended Constraints

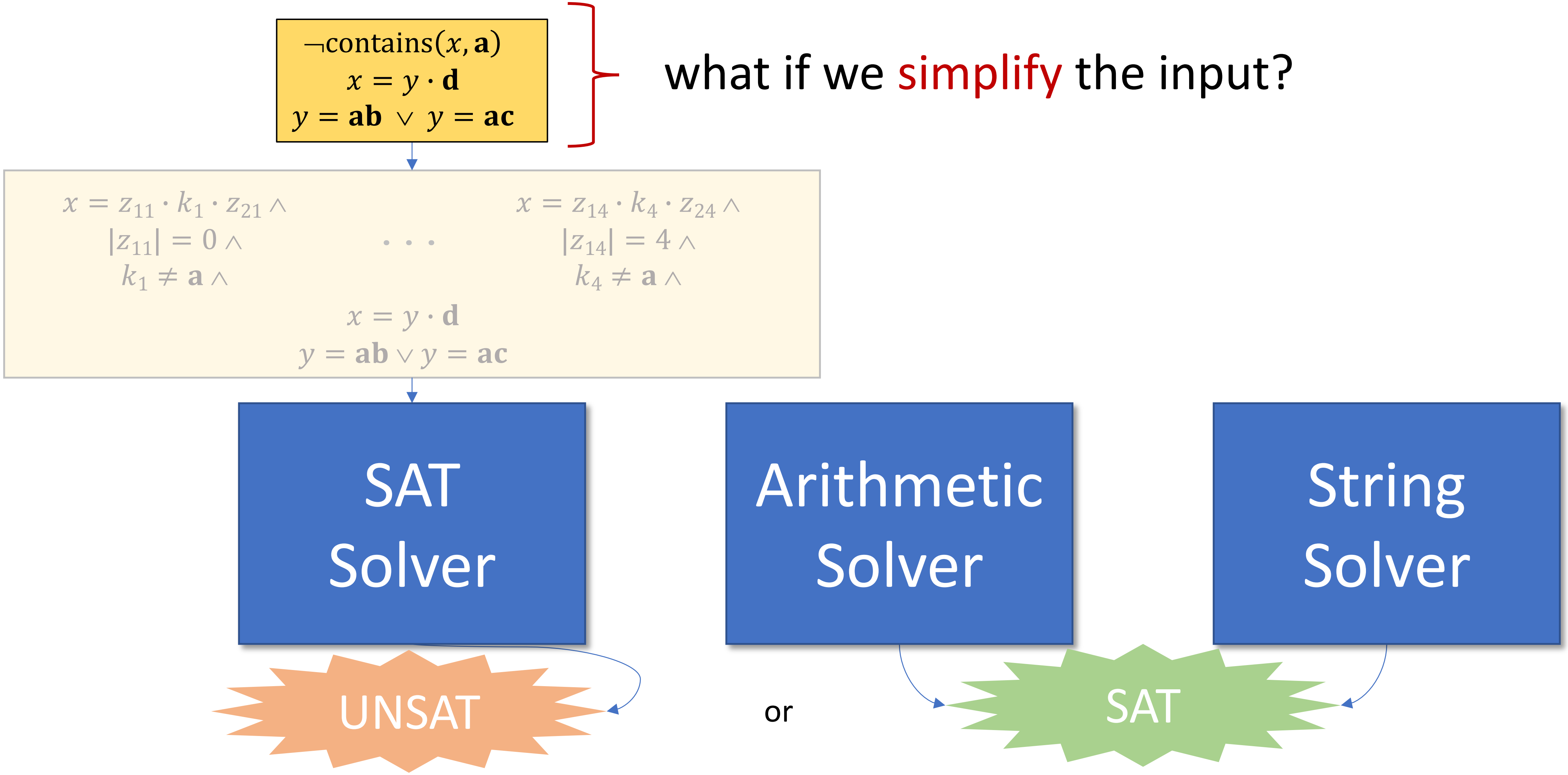


$$\neg\text{contains}(x, \mathbf{a})$$
$$x = y \cdot \mathbf{d}$$
$$y = \mathbf{ab} \lor y = \mathbf{ac}$$

$$x = z_{11} \cdot k_1 \cdot z_{21} \land \quad \cdots \quad x = z_{14} \cdot k_4 \cdot z_{24} \land$$
$$|z_{11}| = 0 \land \quad \cdots \quad |z_{14}| = 4 \land$$
$$k_1 \neq \mathbf{a} \land \quad \cdots \quad k_4 \neq \mathbf{a} \land$$
$$x = y \cdot \mathbf{d}$$
$$y = \mathbf{ab} \lor y = \mathbf{ac}$$

Must deal with a large constraint

(Eager) Expansion of Extended Constraints



$\neg\text{contains}(x, \mathbf{a})$
$x = y \cdot \mathbf{d}$
$y = \mathbf{ab} \vee y = \mathbf{ac}$

what if we simplify the input?

$x = z_{11} \cdot k_1 \cdot z_{21} \wedge$
$|z_{11}| = 0 \wedge$
$k_1 \neq \mathbf{a} \wedge$

$\cdots$

$x = z_{14} \cdot k_4 \cdot z_{24} \wedge$
$|z_{14}| = 4 \wedge$
$k_4 \neq \mathbf{a} \wedge$

$x = y \cdot \mathbf{d}$
$y = \mathbf{ab} \vee y = \mathbf{ac}$

SAT Solver

Arithmetic Solver

String Solver

UNSAT

or

SAT

SMT Solvers + Simplification

All SMT solvers implement simplification techniques

(also called *normalization* or *rewrite* rules)

$$\boxed{\begin{array}{c} \neg\mathrm{contains}(x, \mathbf{a}) \\ x = y \cdot \mathbf{d} \\ y = \mathbf{ab} \lor y = \mathbf{ac} \end{array}}$$

SMT Solvers + Simplification

All SMT solvers implement simplification techniques

(also called *normalization* or *rewrite* rules)

$$\begin{gathered}\neg\text{contains}(x, \mathbf{a}) \\ x = y \cdot \mathbf{d} \\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

$$\downarrow$$

$$\begin{gathered}\neg\text{contains}(y \cdot \mathbf{d}, \mathbf{a}) \\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

$$\downarrow$$

since $x = y \cdot \mathbf{d}$

SMT Solvers + Simplification

All SMT solvers implement simplification techniques

(also called *normalization* or *rewrite* rules)

$$\begin{gathered}\neg\text{contains}(x, \mathbf{a})\\ x = y \cdot \mathbf{d}\\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

$$\downarrow$$

$$\begin{gathered}\neg\text{contains}(y \cdot \mathbf{d}, \mathbf{a})\\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

since $x = y \cdot \mathbf{d}$

$$\downarrow$$

$$\begin{gathered}\neg\text{contains}(y, \mathbf{a})\\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

since $\text{contains}(y \cdot \mathbf{d}, \mathbf{a}) \Leftrightarrow \text{contains}(y, \mathbf{a})$

SMT Solvers + Simplification

All SMT solvers implement simplification techniques

(also called *normalization* or *rewrite* rules)

$$\begin{gathered}\neg\text{contains}(x, \mathbf{a}) \\ x = y \cdot \mathbf{d} \\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

$$\downarrow$$

$$\begin{gathered}\neg\text{contains}(y \cdot \mathbf{d}, \mathbf{a}) \\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

since $x = y \cdot \mathbf{d}$

$$\downarrow$$

$$\begin{gathered}\neg\text{contains}(y, \mathbf{a}) \\ y = \mathbf{ab} \lor y = \mathbf{ac}\end{gathered}$$

since $\text{contains}(y \cdot \mathbf{d}, \mathbf{a}) \Leftrightarrow \text{contains}(y, \mathbf{a})$

- Leads to smaller inputs

  Some problems can be solved by simplification alone

(Lazy) Expansion + Simplification

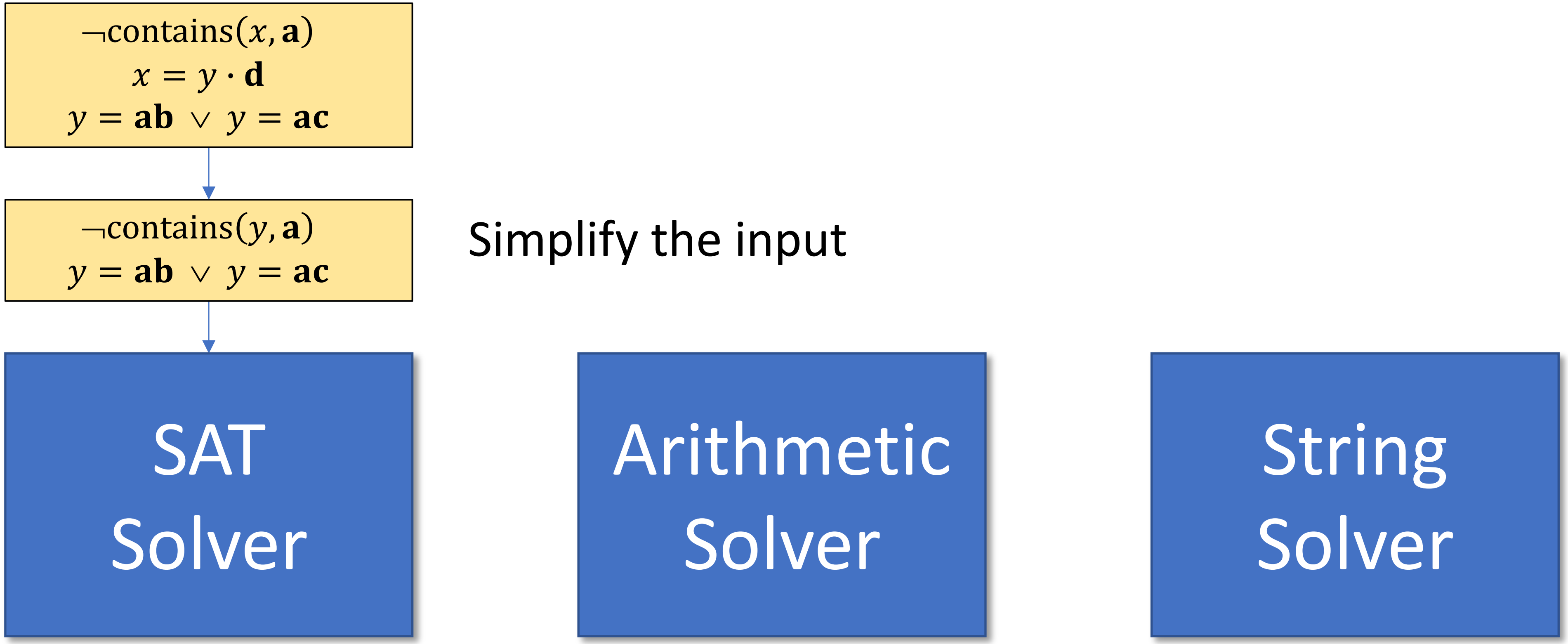
$$\neg\text{contains}(x, \mathbf{a})$$
$$x = y \cdot \mathbf{d}$$
$$y = \mathbf{ab} \lor y = \mathbf{ac}$$

SAT Solver

Arithmetic Solver

String Solver

(Lazy) Expansion + Simplification



$\neg\text{contains}(x, \mathbf{a})$
$x = y \cdot \mathbf{d}$
$y = \mathbf{ab} \lor y = \mathbf{ac}$

$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab} \lor y = \mathbf{ac}$

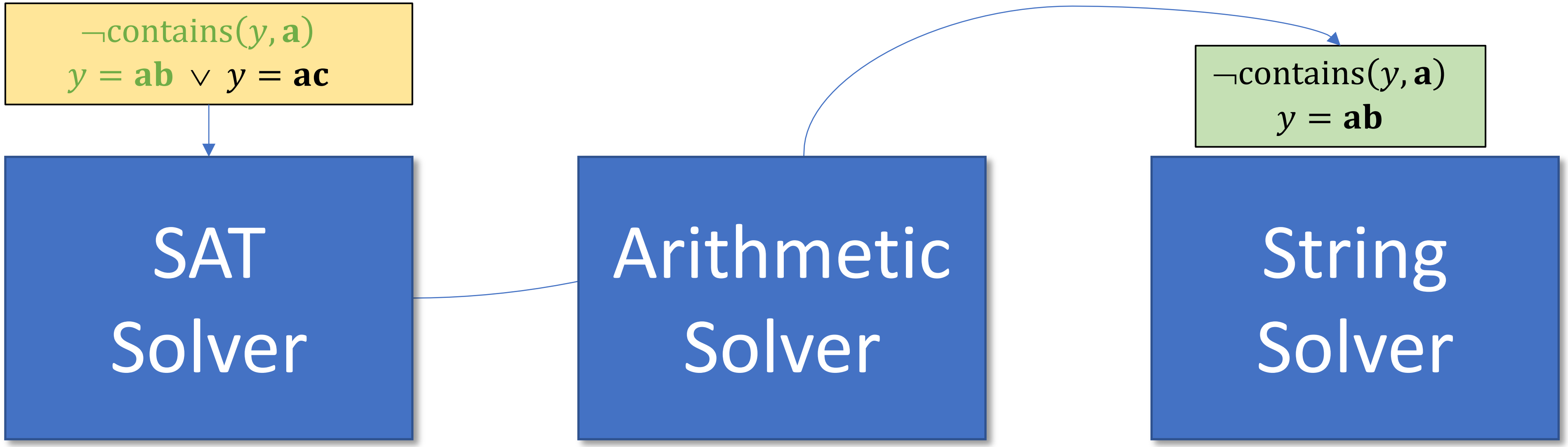
Simplify the input

SAT Solver

Arithmetic Solver

String Solver

(Lazy) Expansion + Simplification



$\neg\mathrm{contains}(y, \mathbf{a})$

$y = \mathbf{ab} \lor y = \mathbf{ac}$

SAT Solver

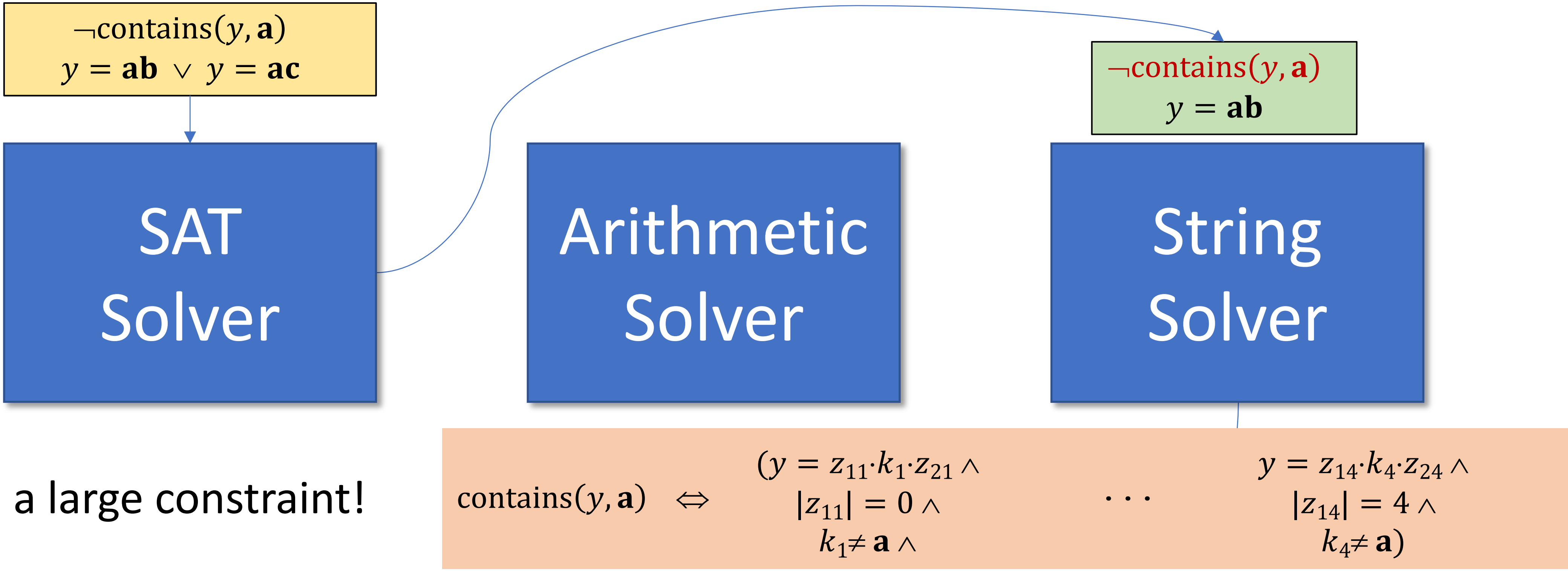
Arithmetic Solver

$\neg\mathrm{contains}(y, \mathbf{a})$

$y = \mathbf{ab}$

String Solver

(Lazy) Expansion + Simplification



$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab} \;\vee\; y = \mathbf{ac}$

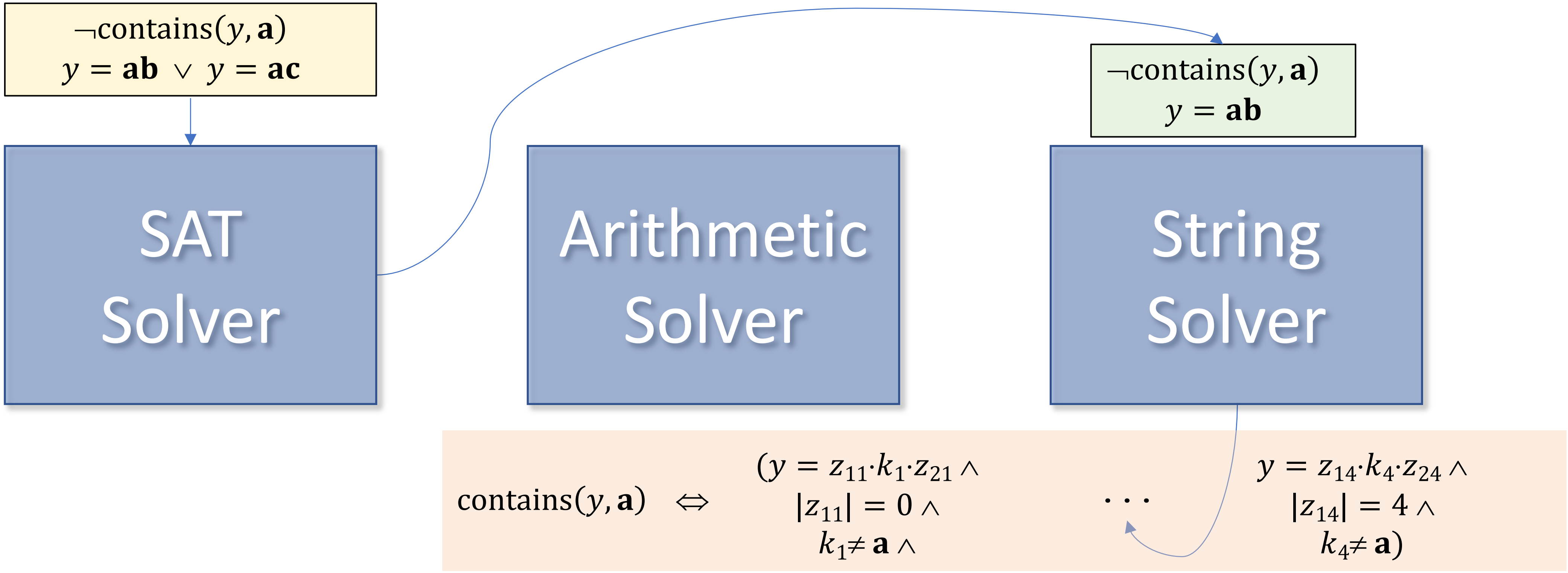
$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab}$

$$\text{contains}(y, \mathbf{a}) \;\Leftrightarrow\; \begin{matrix}(y = z_{11} \cdot k_1 \cdot z_{21} \wedge \\ |z_{11}| = 0 \wedge \\ k_1 \neq \mathbf{a} \wedge\end{matrix} \quad \cdots \quad \begin{matrix}y = z_{14} \cdot k_4 \cdot z_{24} \wedge \\ |z_{14}| = 4 \wedge \\ k_4 \neq \mathbf{a})\end{matrix}$$

Still have a large constraint!

(Lazy) Expansion + Simplification

What if we simplify based on the context?

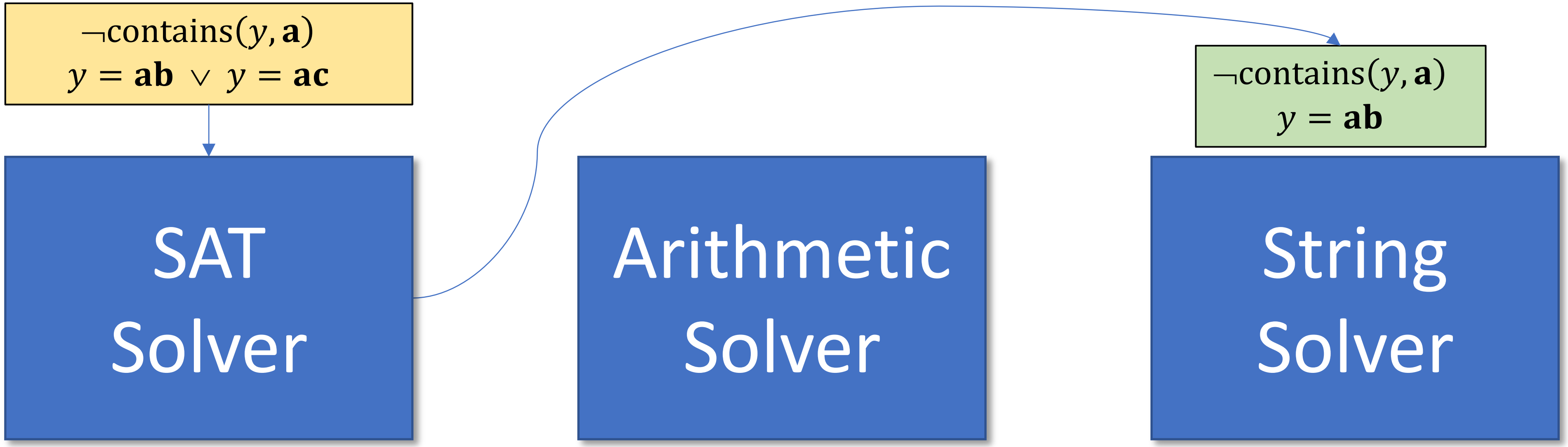


$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab} \lor y = \mathbf{ac}$

$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab}$

$$\text{contains}(y, \mathbf{a}) \Leftrightarrow \begin{array}{c}(y = z_{11} \cdot k_1 \cdot z_{21} \land \\ |z_{11}| = 0 \land \\ k_1 \neq \mathbf{a} \land\end{array} \quad \cdots \quad \begin{array}{c}y = z_{14} \cdot k_4 \cdot z_{24} \land \\ |z_{14}| = 4 \land \\ k_4 \neq \mathbf{a})\end{array}$$

(Lazy) Expansion + Context-Dependent Simplification [Reynolds et al., CAV'17]

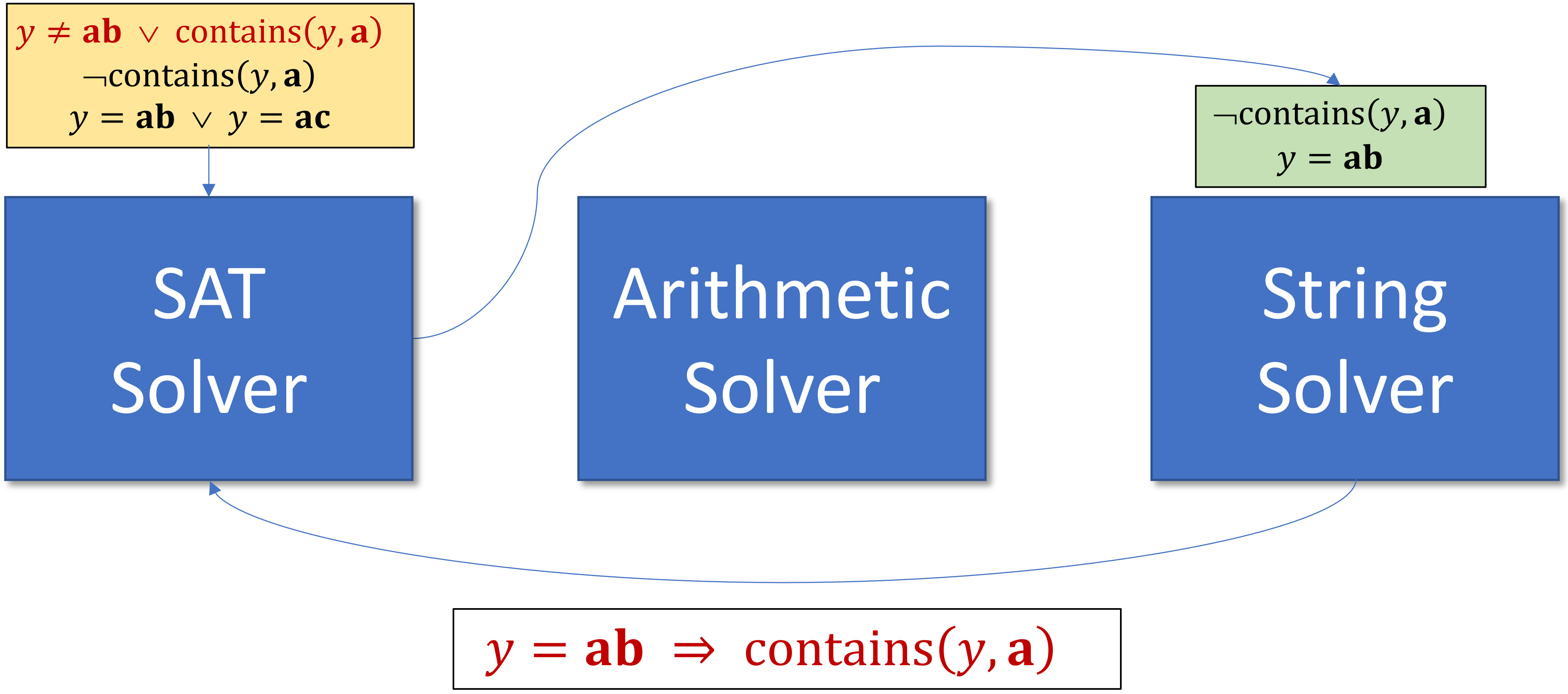


$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab} \lor y = \mathbf{ac}$

$\neg\text{contains}(y, \mathbf{a})$
$y = \mathbf{ab}$

Since $\text{contains}(y, \mathbf{a})$ is true when $y = \mathbf{ab}$ …

(Lazy) Expansion + **Context-Dependent Simplification**



$y \neq \mathbf{ab} \ \vee \ \text{contains}(y, \mathbf{a})$

$\neg\text{contains}(y, \mathbf{a})$

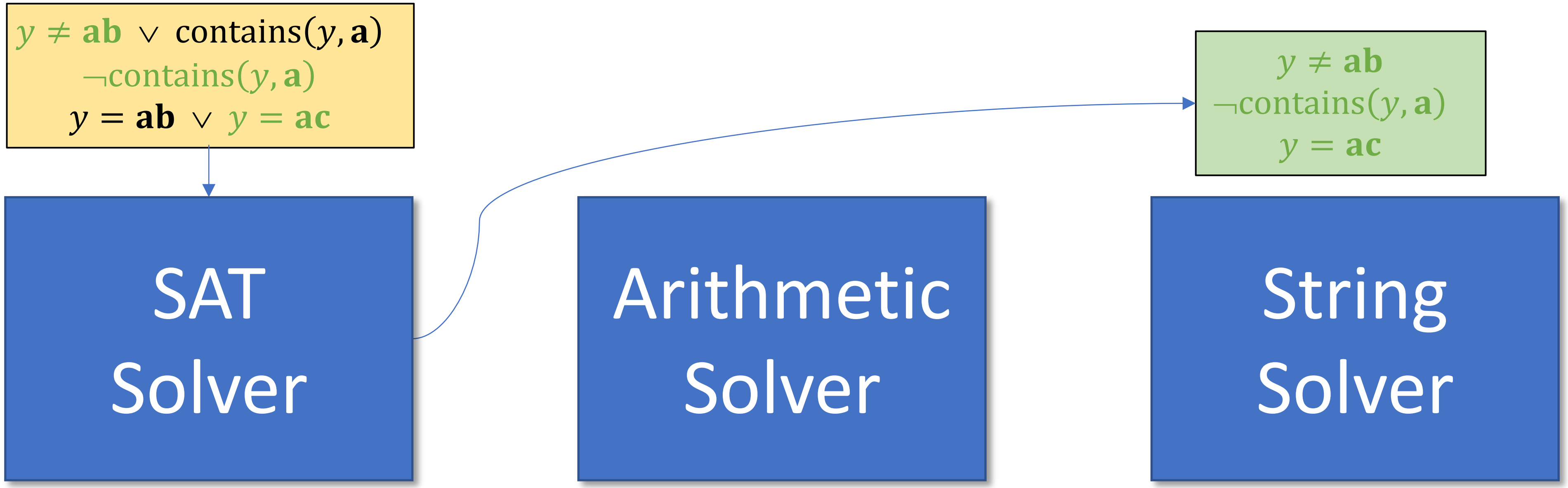
$y = \mathbf{ab} \ \vee \ y = \mathbf{ac}$

$\neg\text{contains}(y, \mathbf{a})$

$y = \mathbf{ab}$

$y = \mathbf{ab} \ \Rightarrow \ \text{contains}(y, \mathbf{a})$

(Lazy) Expansion + **Context-Dependent** Simplification



$y \neq \mathbf{ab} \lor \text{contains}(y, \mathbf{a})$

$\neg\text{contains}(y, \mathbf{a})$

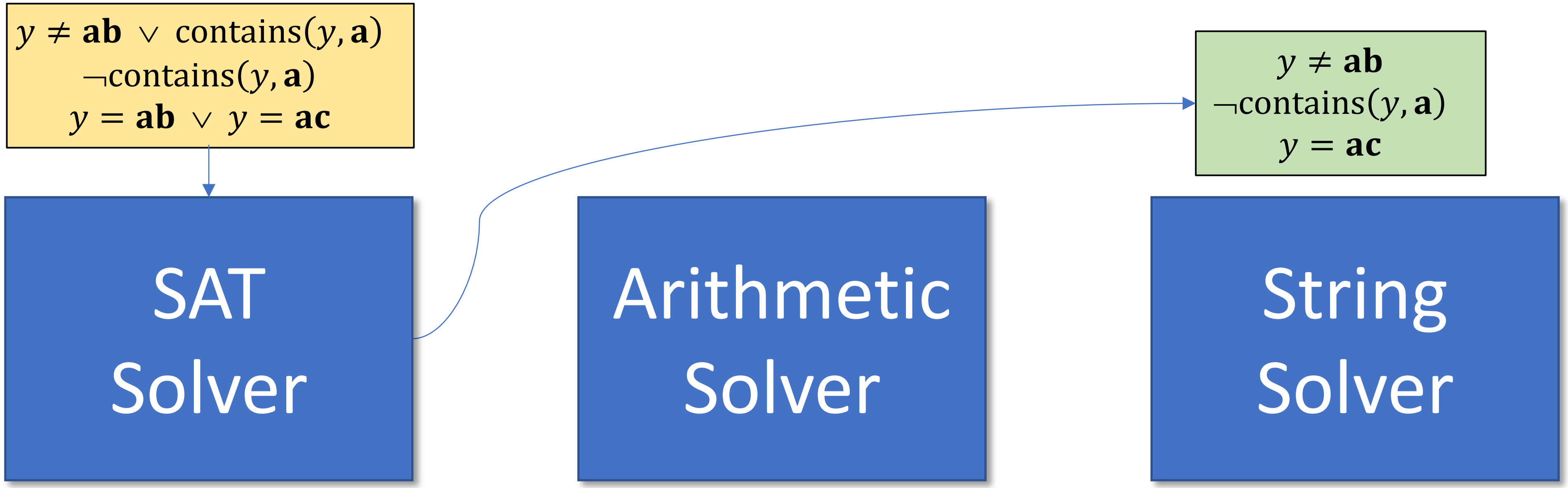
$y = \mathbf{ab} \lor y = \mathbf{ac}$

$y \neq \mathbf{ab}$

$\neg\text{contains}(y, \mathbf{a})$

$y = \mathbf{ac}$

(Lazy) Expansion + **Context-Dependent Simplification**

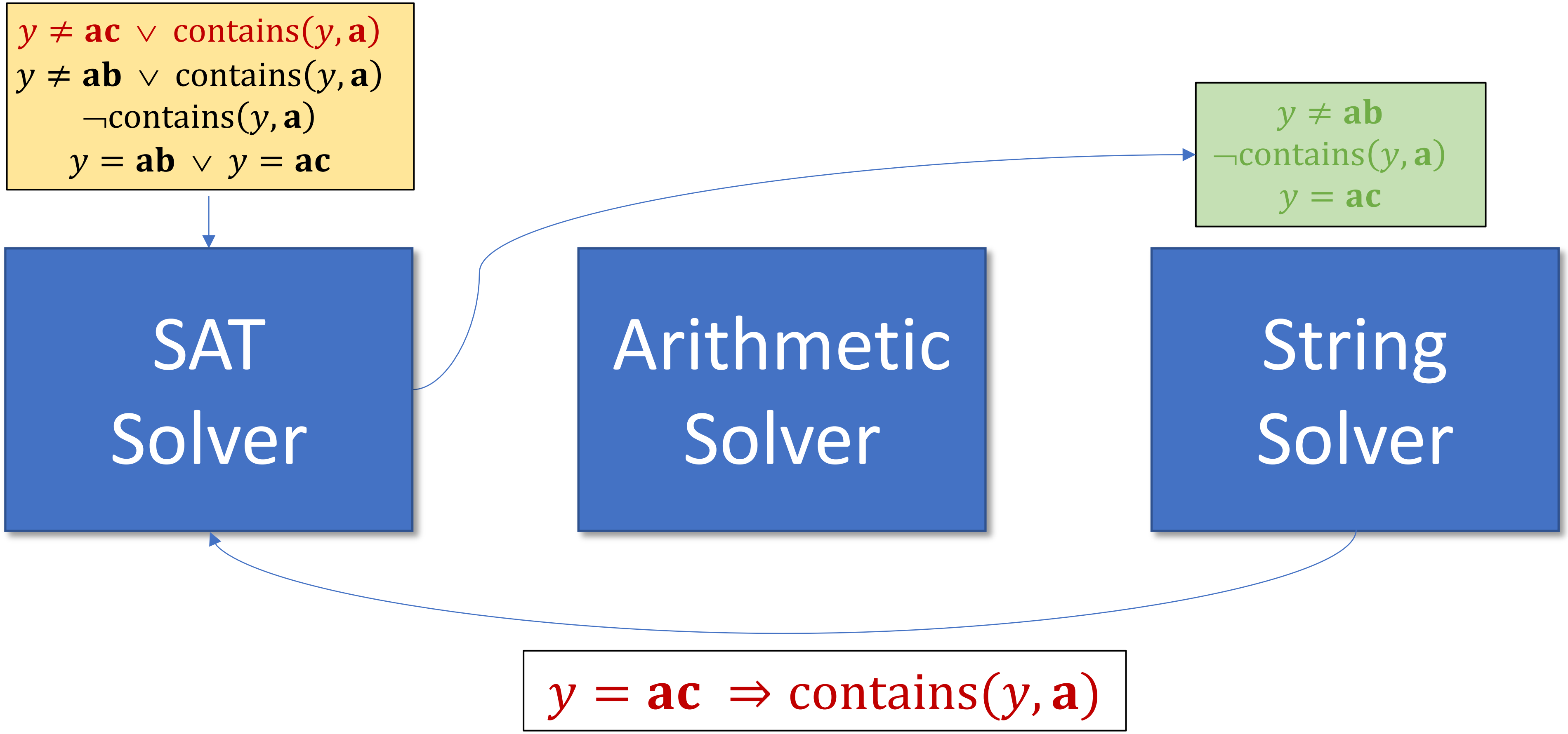


$y \neq \mathbf{ab} \lor \mathrm{contains}(y, \mathbf{a})$
$\neg\mathrm{contains}(y, \mathbf{a})$
$y = \mathbf{ab} \lor y = \mathbf{ac}$

$y \neq \mathbf{ab}$
$\neg\mathrm{contains}(y, \mathbf{a})$
$y = \mathbf{ac}$

$\mathrm{contains}(y, \mathbf{a})$ is true also when $y = \mathbf{ac}$ …

(Lazy) Expansion + **Context-Dependent** Simplification



$y \neq \mathbf{ac} \lor \text{contains}(y, \mathbf{a})$

$y \neq \mathbf{ab} \lor \text{contains}(y, \mathbf{a})$

$\neg\text{contains}(y, \mathbf{a})$

$y = \mathbf{ab} \lor y = \mathbf{ac}$

SAT Solver

Arithmetic Solver

String Solver

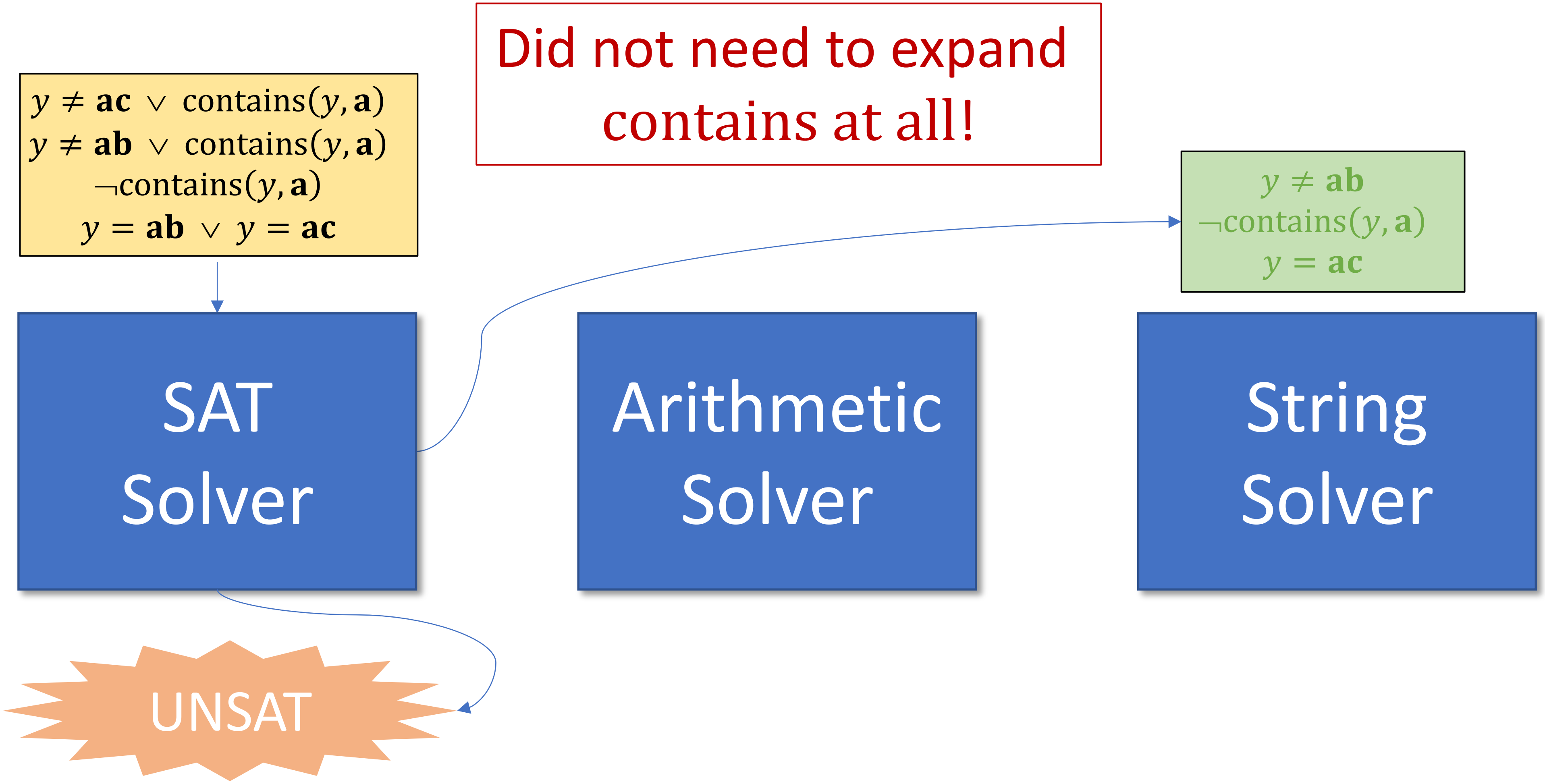
$y \neq \mathbf{ab}$

$\neg\text{contains}(y, \mathbf{a})$

$y = \mathbf{ac}$

$y = \mathbf{ac} \Rightarrow \text{contains}(y, \mathbf{a})$

(Lazy) Expansion + **Context-Dependent** Simplification



Did not need to expand contains at all!

$y \neq \mathbf{ac} \lor \text{contains}(y, \mathbf{a})$

$y \neq \mathbf{ab} \lor \text{contains}(y, \mathbf{a})$

$\neg\text{contains}(y, \mathbf{a})$

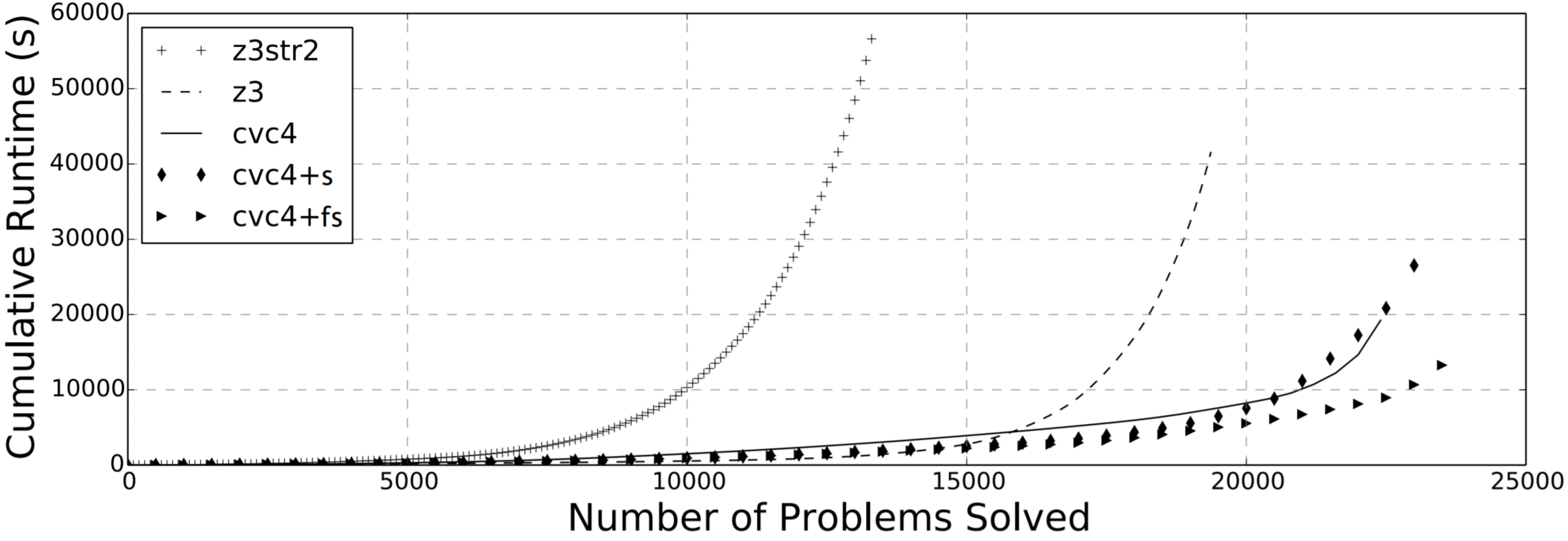
$y = \mathbf{ab} \lor y = \mathbf{ac}$

$y \neq \mathbf{ab}$

$\neg\text{contains}(y, \mathbf{a})$

$y = \mathbf{ac}$

Results on Symbolic Execution [Reynolds et al., CAV'17]



cvc4+fs (context-dependent simplification + finite model finding) solves 23,802 benchmarks in 5h8m

- Without finite model finding, solves 23,266 in 8h46m
- Without either finite model finding or cd-simplification, solves 22,607 in 6h38m

Aggressive Simplifications for Strings

[Reynolds, Noetzli, Tinelli and Barrett, CAV'19]

Many Simplification Rules for Strings

Unlike arithmetic:

$$x + x + 7y = y - 4 \longrightarrow 2x + 6y + 4 = 0$$

… simplification rules for strings can be quite complex:

$$\text{substr}(x \cdot \mathbf{abcd}, 1 + |x|, 2) \longrightarrow \mathbf{bc}$$

$$\text{contains}(\mathbf{abcde}, \mathbf{b} \cdot x \cdot \mathbf{a}) \longrightarrow \bot$$

$$\text{contains}(x \cdot \mathbf{ac} \cdot y, \mathbf{b}) \longrightarrow \text{contains}(x, \mathbf{b}) \vee \text{contains}(y, \mathbf{b})$$

$$\text{indexof}(\mathbf{abc} \cdot x, \mathbf{a} \cdot x, 1) \longrightarrow -1$$

$$\text{replace}(\mathbf{a} \cdot x, \mathbf{b}, y) \longrightarrow \mathbf{a} \cdot \text{replace}(x, \mathbf{b}, y)$$

Abstraction-based Rewriting

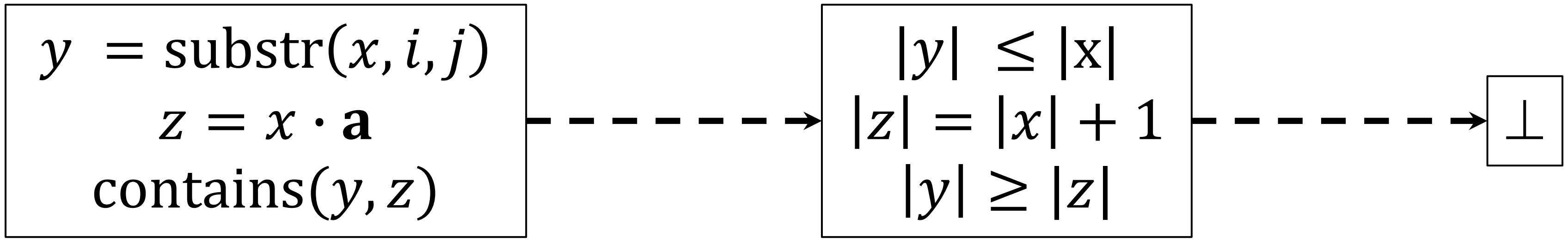
Considering the string containment lattice

$$\boxed{\mathrm{contains}(x \cdot y, \mathrm{substr}(x, i, j))} \longrightarrow \boxed{\top}$$

(since $x \cdot y$ contains x, which contains $\mathrm{substr}(x, \ldots)$)

Abstraction-based Propagators

1. Abstracting strings by their length



$$y = \mathrm{substr}(x, i, j)$$
$$z = x \cdot \mathbf{a}$$
$$\mathrm{contains}(y, z)$$

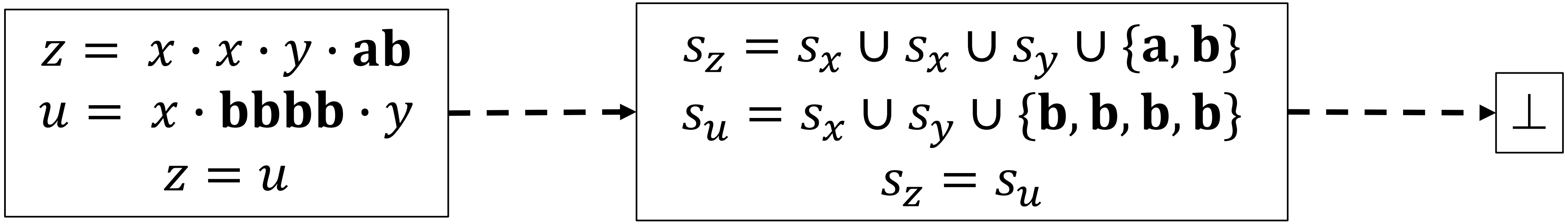
$$|y| \leq |\mathrm{x}|$$
$$|z| = |x| + 1$$
$$|y| \geq |z|$$

$$\bot$$

$$|y| \geq |z| = |x| + 1 \geq |y| + 1 > |y|$$

Abstraction-based Propagators

2. Abstracting strings by their multiset of characters



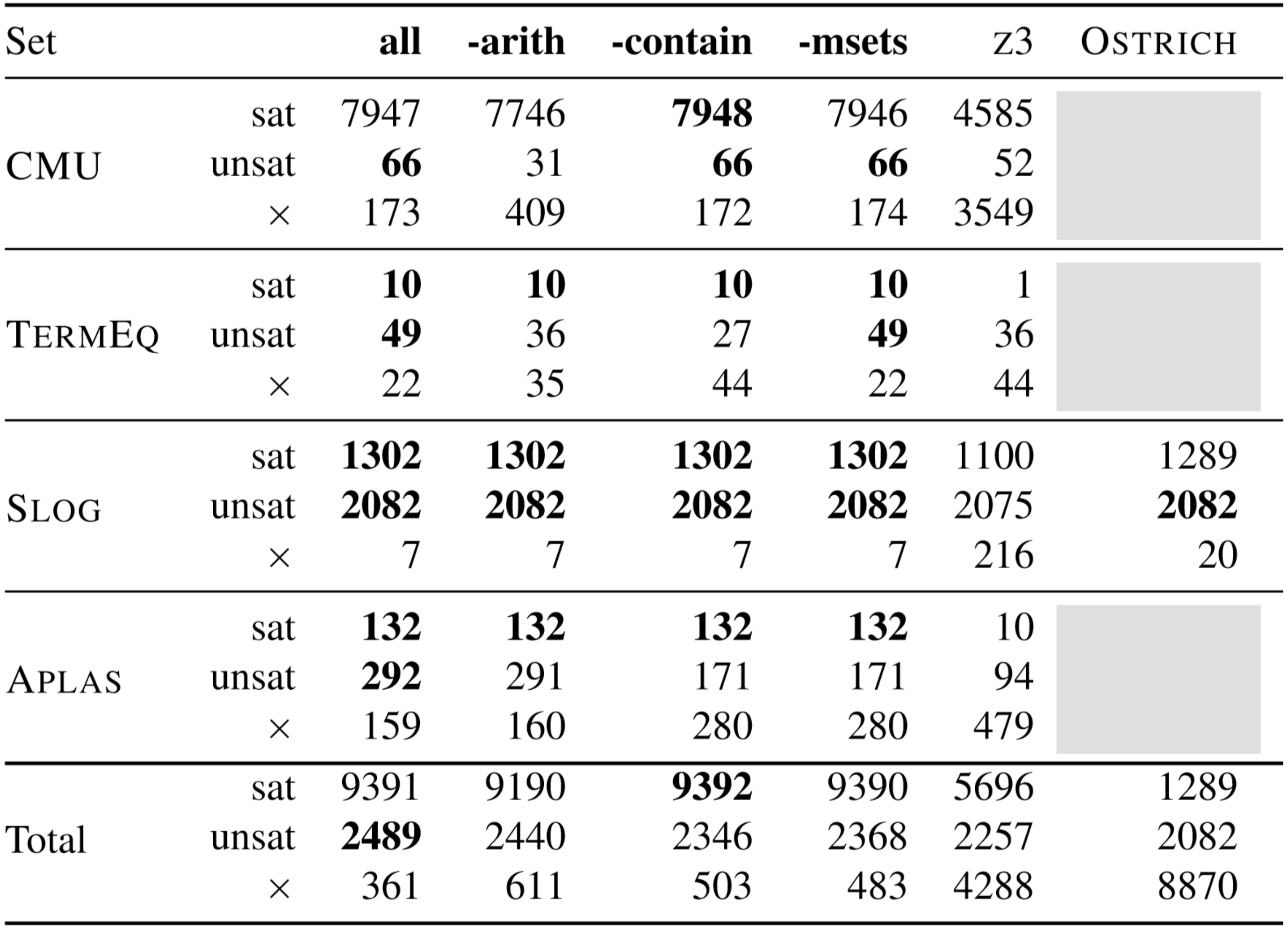
$$\begin{gathered} z = x \cdot x \cdot y \cdot \mathbf{ab} \\ u = x \cdot \mathbf{bbbb} \cdot y \\ z = u \end{gathered}$$

$$\begin{gathered} s_z = s_x \cup s_x \cup s_y \cup \{\mathbf{a}, \mathbf{b}\} \\ s_u = s_x \cup s_y \cup \{\mathbf{b}, \mathbf{b}, \mathbf{b}, \mathbf{b}\} \\ s_z = s_u \end{gathered}$$

$$\boxed{\perp}$$

(s_z contains one extra occurrence of **a** than s_u)

Impact of Aggressive Simplification



Set		all	-arith	-contain	-msets	z3	OSTRICH
CMU	sat	7947	7746	**7948**	7946	4585	
	unsat	**66**	31	**66**	**66**	52	
	×	173	409	172	174	3549	
TERMEQ	sat	**10**	**10**	**10**	**10**	1	
	unsat	**49**	36	27	**49**	36	
	×	22	35	44	22	44	
SLOG	sat	**1302**	**1302**	**1302**	**1302**	1100	1289
	unsat	**2082**	**2082**	**2082**	**2082**	2075	**2082**
	×	7	7	7	7	216	20
APLAS	sat	**132**	**132**	**132**	**132**	10	
	unsat	**292**	291	171	171	94	
	×	159	160	280	280	479	
Total	sat	9391	9190	**9392**	9390	5696	1289
	unsat	**2489**	2440	2346	2368	2257	2082
	×	361	611	503	483	4288	8870

[Reynolds et al., CAV'19]

-arith: w/o arithmetic simplifications
-contain: w/o contain-based simplifications
-mset: w/o multiset-based simplifications

- > 3,000 lines of C++ (and growing) for simplification rules in cvc5
- important aspect of modern string solving

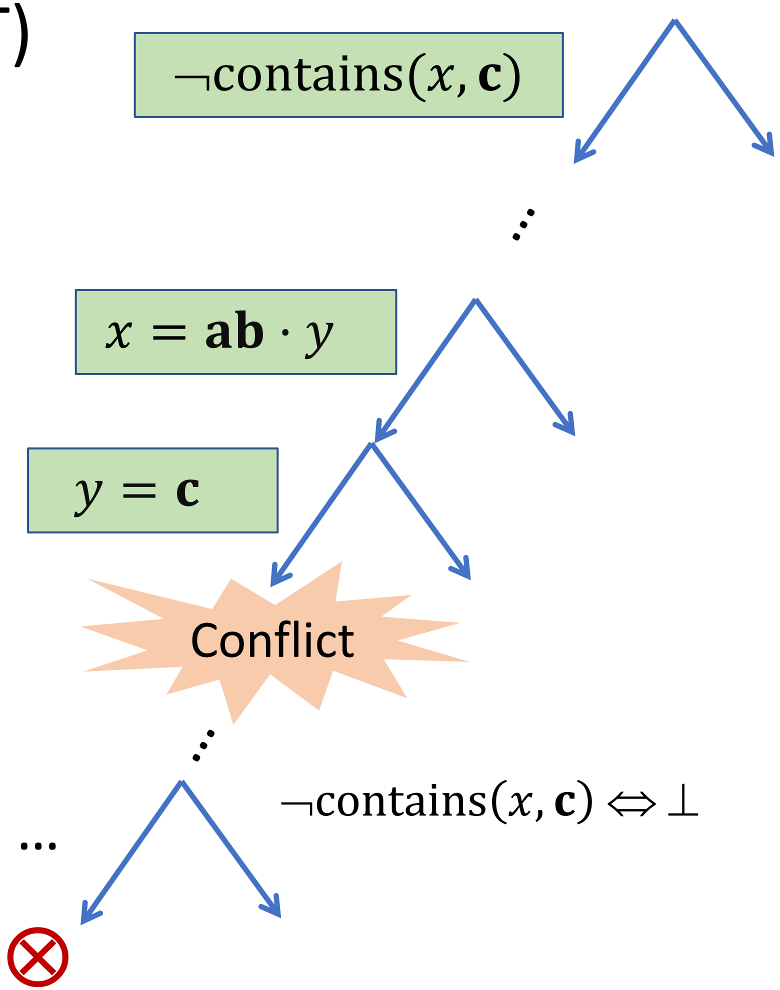
Even Faster Conflicts and Lazier Reductions

[Noetzli, Reynolds, Barbosa, Barrett and Tinelli, CAV’22]

Even Faster Conflicts and Lazier Reductions

Idea: apply simplifications eagerly during CDCL(T) search

- Instrument congruence closure to detect conflicts via:
  - evaluation of concrete terms
  - inferred properties of equivalence classes
    - Upper/lower bounds for integer equivalence classes
    - Prefix and suffix approximations for string equivalence classes
- Report conflicts as soon as they arise
  - Avoids unnecessary expansion of extended functions



$\neg\text{contains}(x, \mathbf{c})$

⋯

$x = \mathbf{ab} \cdot y$

$y = \mathbf{c}$

Conflict

⋯

…

$\neg\text{contains}(x, \mathbf{c}) \Leftrightarrow \bot$

⊗

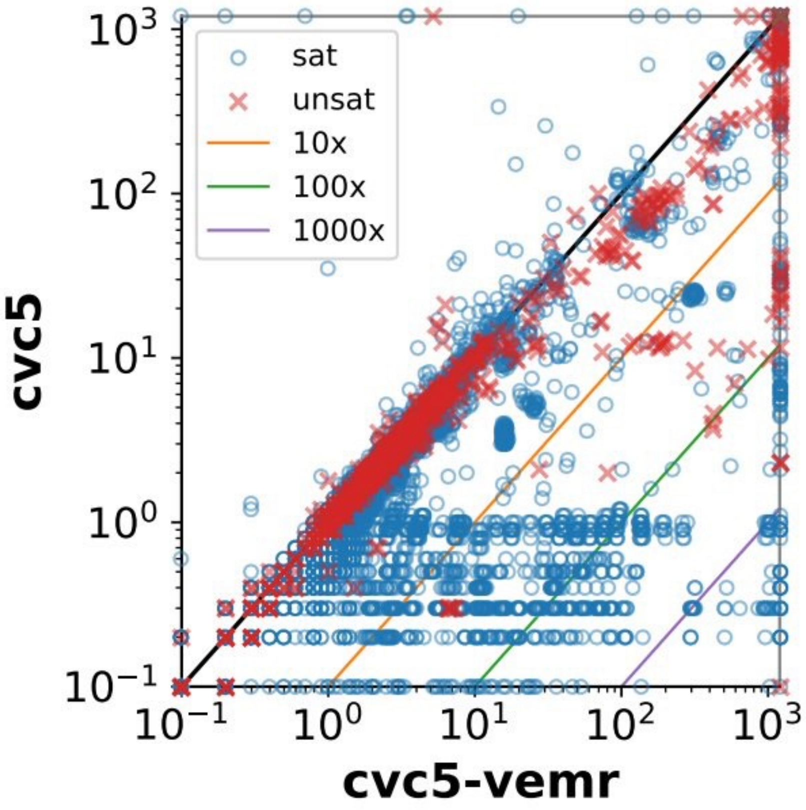
Even Faster Conflicts and Lazier Reductions

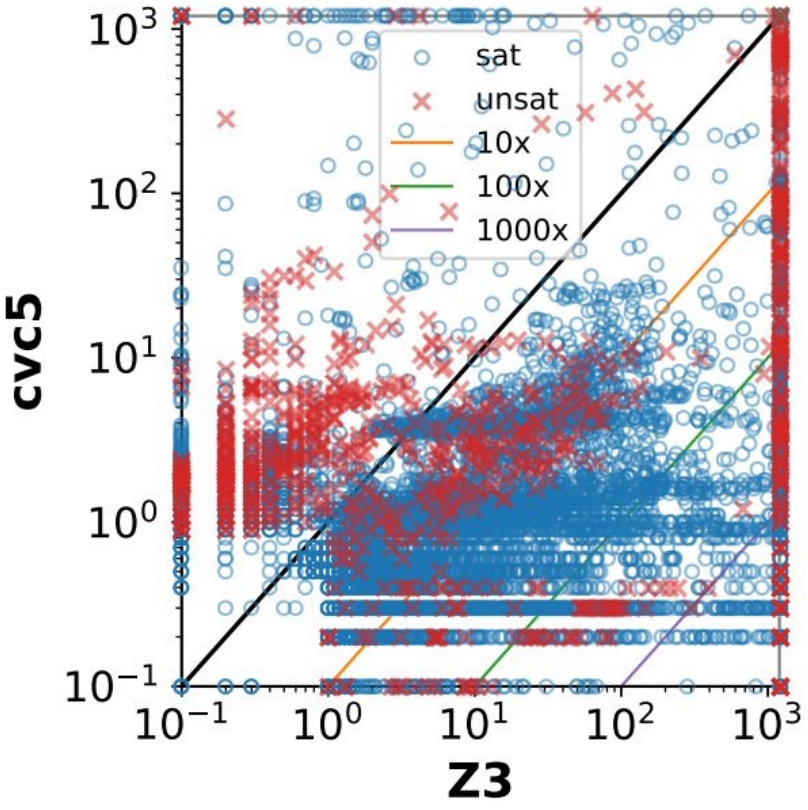
- Avoid reasoning about unnecessary reduction lemmas

- Regular expression inclusion tests
  - ⊗ E.g., do not reduce $x \in \Sigma^*\mathbf{a}\Sigma^*$ if already reduced $x \in \Sigma^*\mathbf{a}\Sigma^*\mathbf{b}\Sigma^*$ to $\top$
    - Since $\mathcal{L}(\Sigma^*\mathbf{a}\Sigma^*\mathbf{b}\Sigma^*) \subseteq \mathcal{L}(\Sigma^*\mathbf{a}\Sigma^*)$
  - Fast incomplete procedure for language inclusion
  - Can also be used for finding conflicts

- Model-based reductions
  - Construct candidate model $\mathcal{M}$
  - ⊗ Do not reduce, e.g., string predicates already satisfied by $\mathcal{M}$
  - Often, negative RE membership predicates are satisfied by current model

Even Faster Conflicts and Lazier Reductions





Results on 10,857 SMT-LIB string benchmarks; 1,200s timeout

- cvc5 solves 10,347; z3 solves 8,863

Witness Sharing + RE Elim

[Reynolds, Noetzli, Tinelli and Barrett, FMCAD'20]

Witness Sharing

Observation:

- There are often equivalent ways of expressing the same thing
  - E.g., string y is the result of removing the first character from string x:

$$\exists z.\ x = z \cdot y \ \wedge\ |z| = 1 \qquad \mathrm{substr}(x, 1, |x| - 1) = y \qquad x \in \Sigma \cdot y$$

- Solving word equations, extended functions, and REs introduces many fresh variables

Idea:

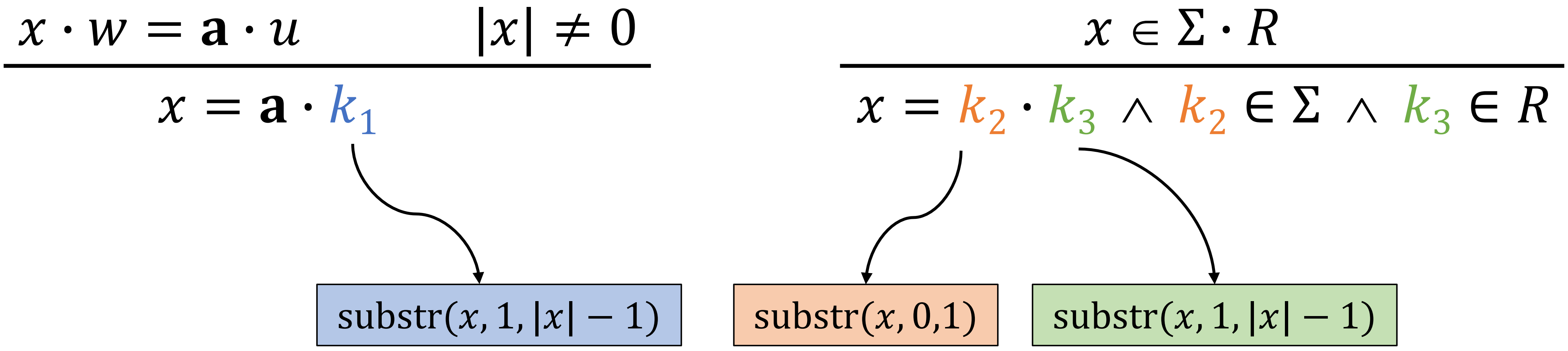
- Formalize the definition for each introduced variable's *witness form*
- Reuse variables whose witness forms are semantically equivalent

Witness Sharing (Example)

$$\frac{x \cdot w = \mathbf{a} \cdot u \qquad |x| \neq 0}{x = \mathbf{a} \cdot k_1}$$

$$\frac{x \in \Sigma \cdot R}{x = k_2 \cdot k_3 \ \wedge \ k_2 \in \Sigma \ \wedge \ k_3 \in R}$$

Witness Sharing (Example)



$$\frac{x \cdot w = \mathbf{a} \cdot u \qquad |x| \neq 0}{x = \mathbf{a} \cdot k_1}$$

$$\frac{x \in \Sigma \cdot R}{x = k_2 \cdot k_3 \wedge k_2 \in \Sigma \wedge k_3 \in R}$$

$\mathrm{substr}(x, 1, |x| - 1)$

$\mathrm{substr}(x, 0, 1)$

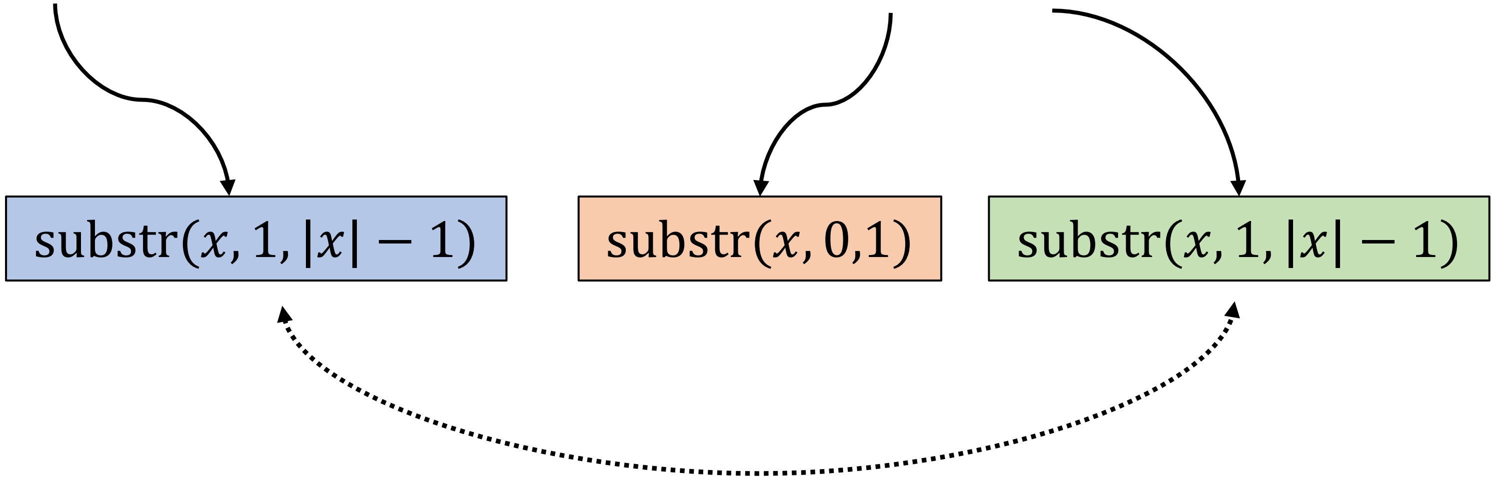
$\mathrm{substr}(x, 1, |x| - 1)$

witness forms

Witness Sharing (Example)

$$\frac{x \cdot w = \mathbf{a} \cdot u \qquad |x| \neq 0}{x = \mathbf{a} \cdot k_1}$$

$$\frac{x \in \Sigma \cdot R}{x = k_2 \cdot k_3 \wedge k_2 \in \Sigma \wedge k_3 \in R}$$



$\mathrm{substr}(x, 1, |x| - 1)$

$\mathrm{substr}(x, 0,1)$

$\mathrm{substr}(x, 1, |x| - 1)$

Reuse variables whose witness form are (semantically) equivalent

$\Rightarrow$ *Can use aggressive simplification to detect equivalent witness forms*

Regular Expression Elimination

Idea: reduce REs to extended string constraints

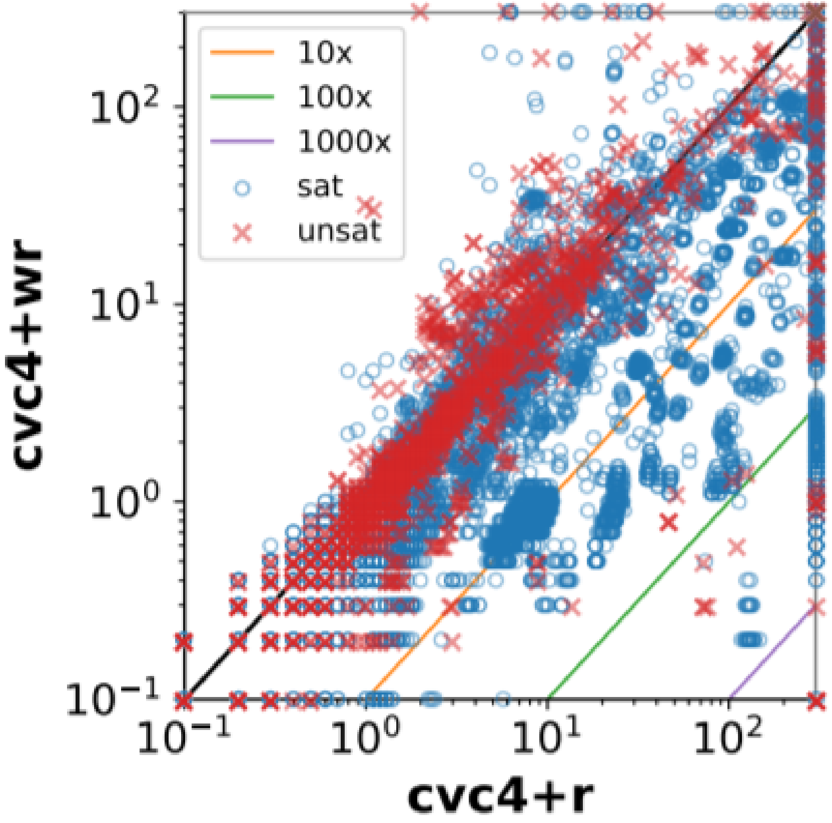
- Possible for many RE memberships occurring in practice

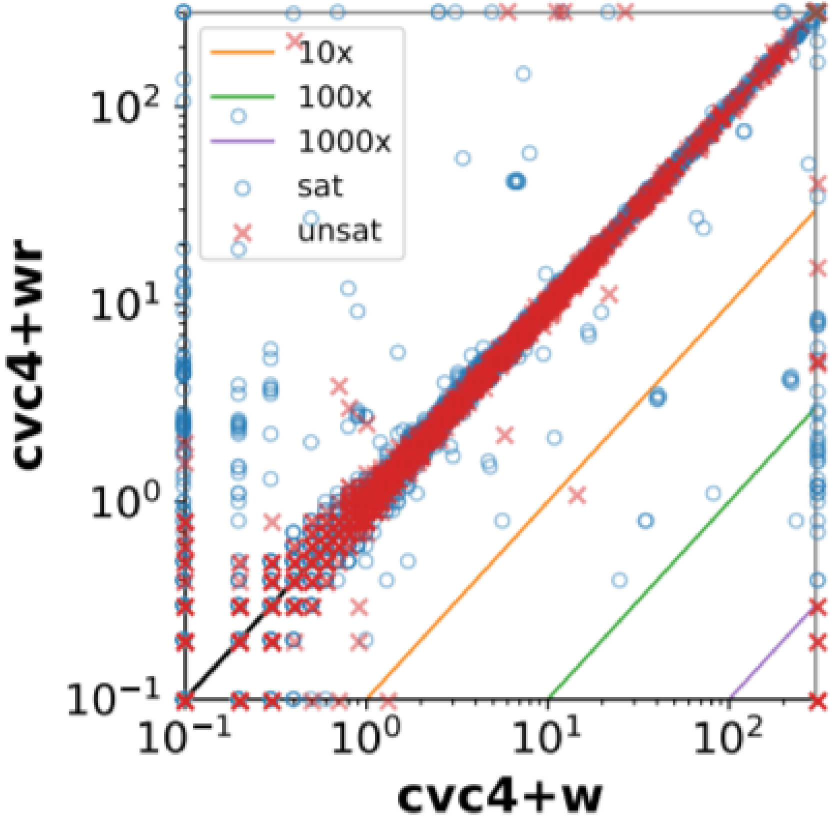
$$\boxed{x \in \Sigma\Sigma^*\Sigma} \quad \Longleftrightarrow \quad \boxed{|x| \geq 2}$$

$$\boxed{x \in \Sigma^*\mathbf{abc}\Sigma^*} \quad \Longleftrightarrow \quad \boxed{\text{contains}(x, \mathbf{abc})}$$

$$\boxed{x \in \Sigma^*\mathbf{a}\Sigma^*\mathbf{bcd}\Sigma^*} \quad \Longleftrightarrow \quad \boxed{\begin{array}{c}\text{contains}(x, \mathbf{a}) \ \wedge \\ \text{contains}(\text{substr}(x, \text{indexof}(x, \mathbf{a}, 1) + 1, |x|), \mathbf{bcd})\end{array}}$$

Impact of Witness Sharing + RE elim





String to Code Point Conversion

[Reynolds, Noetzli, Tinelli and Barrett, IJCAR’20]

Adding string-to-code operator **code**

Assume ordering on characters of alphabet Σ of size n:

- $c_1 < \cdots < c_n$
- For each character c_i, we call i its *code point*

$\mathrm{code} : \mathrm{String} \to \mathrm{Int}$ is defined as follows:

1. $\mathrm{code}(c_i) = i$ for all $c_i \in \Sigma$
2. $\mathrm{code}(w) = -1$ for all $\mathrm{w} \in \Sigma^+$

Fragment with string length + code points (w/o concatenation):

- Devised a solving procedure that is sound, complete, and terminating

Reductions: Conversion Functions

Using code leads to efficient reductions, including:

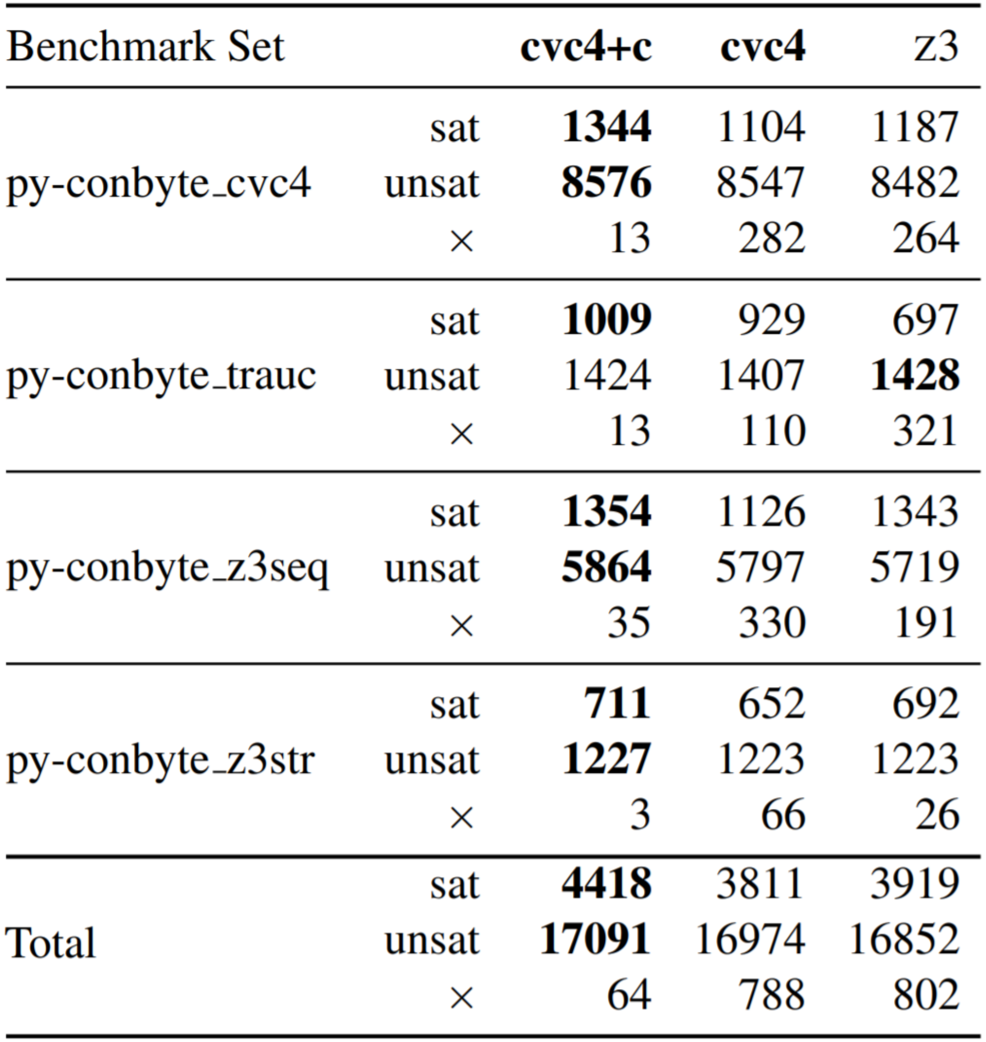
- Conversion between strings and integers toInt:
  - ⊗ … $\mathrm{ite}(x[i] = \mathbf{9}, 9, \mathrm{ite}(x[i] = \mathbf{8}, 8, \ldots\ \mathrm{ite}(x[i] = \mathbf{0}, 0, -1) \ldots)$
  - $\Rightarrow$ … $\mathrm{ite}(48 \leq \mathrm{code}(x[i]) \leq 57, \mathrm{code}(x[i]) - 48, -1)$
- Conversion between lowercase and uppercase strings toLower:
  - ⊗ … $\mathrm{ite}(x[i] = \mathbf{A}, \mathbf{a}, \mathrm{ite}(x[i] = \mathbf{B}, \mathbf{b}, \ldots \mathrm{ite}(x[i] = \mathbf{Z}, \mathbf{z}, x[i]) \ldots)$
  - $\Rightarrow$ … $\mathrm{code}(x[i]) + \mathrm{ite}(65 \leq \mathrm{code}(x[i]) \leq 90, 32, 0)$

Reductions: Conversion Functions

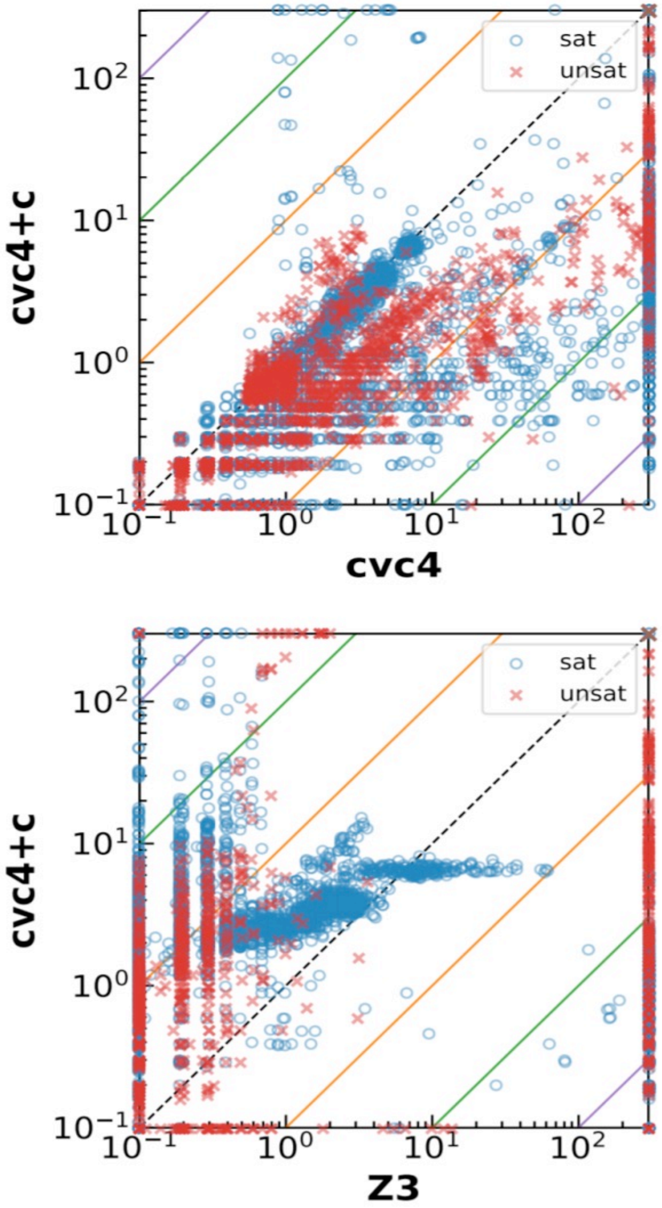
Using code leads to efficient reductions, including:

- Lexicographic ordering:
  - ⊗ $x \leq y \Leftrightarrow \exists i \; \ldots \; (x[i] = y[i] \vee (x[i] = \mathbf{a} \wedge y[i] = \mathbf{b}) \vee (x[i] = \mathbf{a} \wedge y[i] = \mathbf{c}) \; \ldots)$
  - ⇒ $x \leq y \Leftrightarrow \exists i \; \ldots \mathrm{code}(x[i]) \leq \mathrm{code}(y[i])$
- Regular expression ranges:
  - ⊗ $x \in \mathrm{range}(c_1, c_2) \Leftrightarrow |x| = 1 \wedge (x = c_1 \vee \cdots \vee x = c_2)$
  - ⇒ $x \in \mathrm{range}(c_1, c_2) \Leftrightarrow \mathrm{code}(c_1) \leq \mathrm{code}(x) \leq \mathrm{code}(c_2)$

Experimental Results



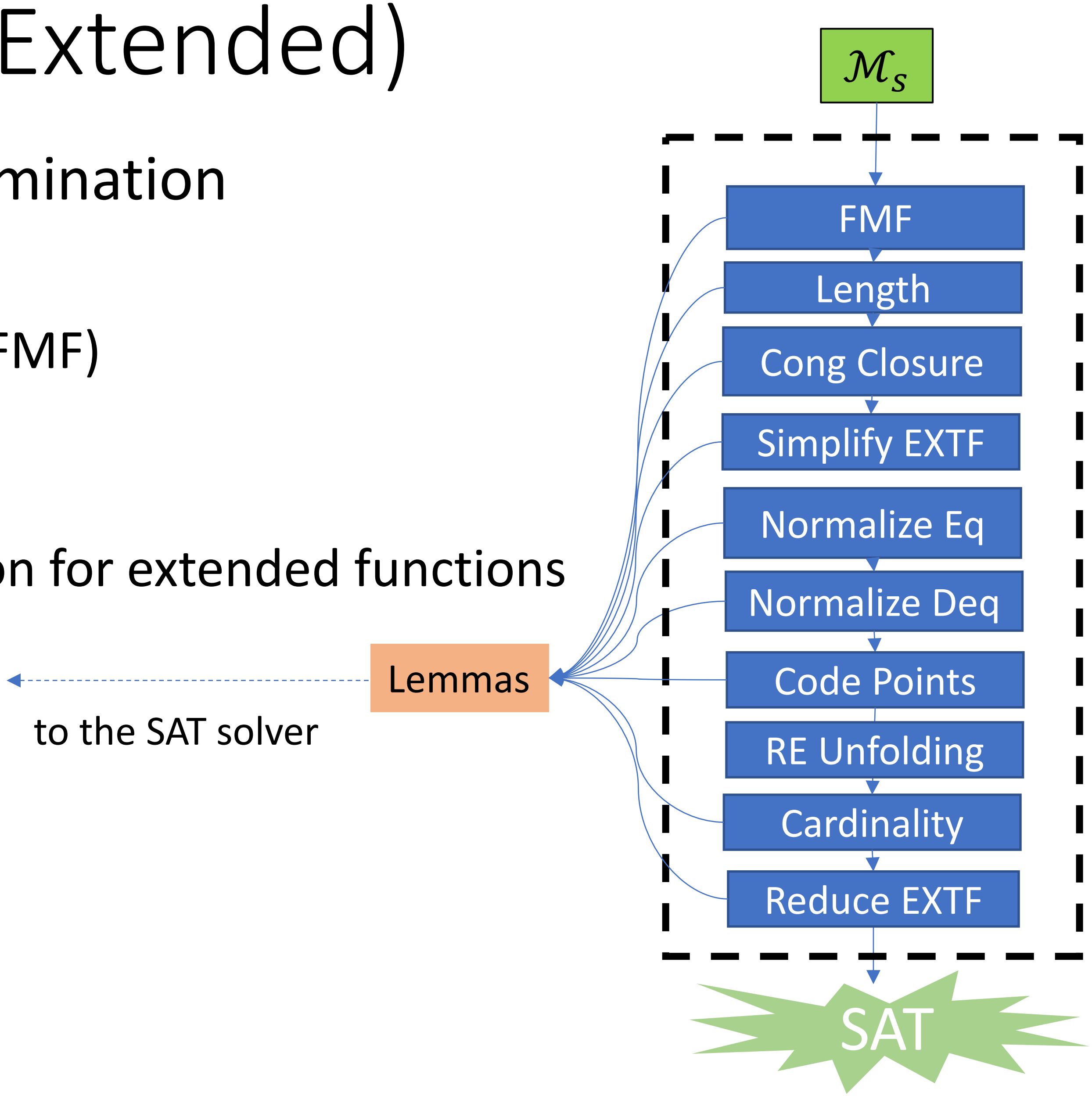
Benchmark Set		**cvc4+c**	**cvc4**	z3
py-conbyte_cvc4	sat	**1344**	1104	1187
	unsat	**8576**	8547	8482
	×	13	282	264
py-conbyte_trauc	sat	**1009**	929	697
	unsat	1424	1407	**1428**
	×	13	110	321
py-conbyte_z3seq	sat	**1354**	1126	1343
	unsat	**5864**	5797	5719
	×	35	330	191
py-conbyte_z3str	sat	**711**	652	692
	unsat	**1227**	1223	1223
	×	3	66	26
Total	sat	**4418**	3811	3919
	unsat	**17091**	16974	16852
	×	64	788	802



- 10x t/o reduction
- Faster runtimes
- Improvement wrt state of the art

String Theory Solver (Extended)

- Preprocess based on reg-exp elimination
- Then, run inference strategy:
  1. Split on sum of lengths bound (FMF)
  2. Elaborate length constraints
  3. Congruence closure
  4. Context-dependent simplification for extended functions
  5. Normalize string equalities
  6. Normalize string disequalities
  7. Subprocedure for code points
  8. Regular expression unfolding
  9. Check cardinality constraints
  10. Reduce extended functions



Conclusions

SMT solvers can provide:

  - Efficient (incomplete) procedure for word equations with length
  - FMF, context-dependent simplification, RE elimination, witness sharing, …

Ongoing work in cvc5:

  - Proofs and proof certificates
  - Array-like reasoning (update + slices)

THE UNIVERSITY
OF IOWA

Stanford
University

- cvc5 is open-source, available at https://cvc5.github.io/
  - Also supports theory of sequences, further extensions

Thanks for listening!