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

Tianyi Liang
Andrew Reynolds
Cesare Tinelli
Morgan Deters
Clark Barrett

University of Iowa
New York University
Motivation: Security Applications

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';
    }
} else {
    cout << "Wrong Password";
}
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 (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"))
Objectives

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

- Theoretical complexity of:
  - Word equation problem: \( \text{PSPACE} \)
  - ...with length constraints: \( \text{OPEN} \)
  - ...with other functions (e.g. \( \text{replace} \)): \( \text{UNDECIDABLE} \)
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 $\Sigma^*$ (a, ab, cbc...) 
  – Free constants or “variables” (x, y, z, w...) 
  – String concatenation
    
    $\_ \cdot \_ : \text{String} \times \text{String} \rightarrow \text{String}$
  – Length terms
    
    $\text{len}(\_ ) : \text{String} \rightarrow \text{Int}$

• Example input:

$$\text{len}(x) > \text{len}(y) \land (x \cdot b = y \cdot ab \lor x = y)$$
DPLL(T): Find Satisfying Assignment

\[ \text{len}(x) > \text{len}(y) \]

\[(x \cdot b = y \cdot ab \lor x = y)\]

\text{SAT Solver}

Find \text{satisfying assignment}

\[ \text{len}(x) > \text{len}(y) \]

\[ x \cdot b = y \cdot ab \]

\[ : \]
DPLL(T): Cooperating *Theory Solvers*

\[ \text{len}(x) > \text{len}(y) \]
\[ x \cdot b = y \cdot ab \]

Purify and distribute constraints to corresponding theory solvers

- **Theory LIA**
  - \[ \text{len}(x) > \text{len}(y) \]

- **Theory Strings**
  - \[ x \cdot z = y \cdot ab \]
DPLL(T): Cooperating Theory Solvers

Communication (dis)equalities over shared terms

[Nelson-Oppen]

Theory

LIA

len(x) > len(y)

Strings

len(x) \neq len(y)

x \cdot z = y \cdot ab

len(x) \neq len(y)
DPLL(T): Cooperating Theory Solvers

- Theory
  - LIA

- Strings
  - $\text{len}(x) > \text{len}(y)$
  - $x \cdot z = y \cdot ab$
  - $\text{len}(x) \neq \text{len}(y)$
Summary of Approach

• Approach for $A \cup S \cup L$ in four steps:

1. Check arithmetic constraints $A$
2. Normalize equalities in $S$
3. Normalize disequalities in $S$
4. Check cardinality of $\Sigma$
Check Length Constraints

- Add **equalities** to $A$ based on terms from $S$

Theory
- LIA

Theory
- Strings

$A$
- $\text{len}(z) > \text{len}(w)$

$S$
- $x \cdot z = y \cdot w \cdot ab$

$L$
- $\text{len}(x) = \text{len}(y)$
- $\text{len}(z) \neq \text{len}(w)$

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$
Check Length Constraints

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

Theory
LIA

$\text{len}(z) > \text{len}(w)$
$\text{len}(x) + \text{len}(z) = \text{len}(y) + \text{len}(w) + 2$

$A \\implies \text{Check if } A \text{ is satisfiable}$

Theory
Strings

$x \cdot z = y \cdot w \cdot ab$

$\text{len}(x) = \text{len}(y)$
$\text{len}(z) \neq \text{len}(w)$

$S$

$L$
Normalize Equalities

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

\[ S \quad \frac{x \cdot z = y \cdot w \cdot ab}{L} \quad \begin{cases} \text{len}(x) = \text{len}(y) \\ \text{len}(z) \neq \text{len}(w) \end{cases} \]
Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of \( \Sigma \)

\[
\begin{align*}
S & \quad \xcdot z = y \cdot w \cdot ab \\
\text{I} & \quad \text{len}(x) = \text{len}(y) \\
& \quad \text{len}(z) \neq \text{len}(w)
\end{align*}
\]
Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

$S$

\[
x \cdot z = y \cdot w \cdot ab \\
x = y
\]

$\not= \text{Propagate, since } \text{len}(x) = \text{len}(y) \text{ and } \text{len}(z) \not= \text{len}(w)$

$x \quad z$

$y \\ w \\ ab$

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$
Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

$S$

\[
\begin{cases}
x \cdot z = y \cdot w \cdot ab \\
x = y
\end{cases}
\]

$L$

\[
\begin{cases}
\text{len}(x) = \text{len}(y) \\
\text{len}(z) \neq \text{len}(w)
\end{cases}
\]

We have that \(\text{len}(z) \neq \text{len}(w)\)

$\Sigma$

$\prod$

\[\text{We have that } \text{len}(z) \neq \text{len}(w)\]

$\Sigma$

$\prod$

\[\text{We have that } \text{len}(z) \neq \text{len}(w)\]
Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

$\Sigma$

$$x \cdot z = y \cdot w \cdot ab$$

$$x = y$$

$$z = w \cdot z'$$

$\mathbb{L}$

$$\text{len}(x) = \text{len}(y)$$

$$\text{len}(z) \neq \text{len}(w)$$

$X$

$Z$

$w$

$z'$

$Y$

$w$

$ab$
Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

$\Sigma$

- $x \cdot z = y \cdot w \cdot ab$
- $x = y$
- $z = w \cdot z'$

$L$

- $\text{len}(x) = \text{len}(y)$
- $\text{len}(z) \neq \text{len}(w)$

Diagram:

- $X \quad Z$
- $W \quad z'$
- $y \quad w \quad ab$

II Reflexive
Normalize Equalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

\[
\begin{align*}
S & \quad \{ x \cdot z = y \cdot w \cdot ab, \quad x = y, \quad z = w \cdot z', \quad z' = ab \} \\
\end{align*}
\]

\[
\begin{align*}
\mathbb{I} & \quad \{ \text{len}(x) = \text{len}(y), \quad \text{len}(z) \neq \text{len}(w) \} \\
\end{align*}
\]

\[
\begin{align*}
\Rightarrow \text{Normal form} \\
\end{align*}
\]
Normalize Disequalities

- Disequalities normalized analogously

\[ S \ni x \cdot z \cdot z' \neq y \cdot w \cdot ab \]
\[ \Rightarrow x = y \]

\[ \begin{align*}
\text{1. Check length constraints} \\
\text{2. Normalize equalities} \\
\text{3. Normalize disequalities} \\
\text{4. Check cardinality of } S
\end{align*} \]

\[ \begin{align*}
\text{len}(x) &= \text{len}(y) \\
\text{len}(z) &= \text{len}(w)
\end{align*} \]
Normalize Disequalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

Given $x \cdot z \cdot z' \neq y \cdot w \cdot ab$

$x = y$

$\text{len}(x) = \text{len}(y)$
$\text{len}(z) = \text{len}(w)$

$S$

$\Pi$ Given

$x$ $z$ $z'$

$y$ $w$ $ab$
Normalize Disequalities

1. Check length constraints
2. Normalize equalities
3. Normalize disequalities
4. Check cardinality of $\Sigma$

\[ S \]
\[
\begin{align*}
x \cdot z \cdot z' & \neq y \cdot w \cdot ab \\
x &= y \\
z & \neq w
\end{align*}
\]

\[ L \]
\[
\begin{align*}
\text{len}(x) &= \text{len}(y) \\
\text{len}(z) &= \text{len}(w)
\end{align*}
\]

\[
\begin{array}{c}
x & \quad z & \quad z' \\
\downarrow & \downarrow & \downarrow \\
\downarrow & \downarrow & \downarrow \\
y & \quad w & \quad ab
\end{array}
\]

$\Rightarrow$ Normal form
Check Cardinality of $\Sigma$

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

Then:
  - $\Sigma$ is unsatisfiable
- Performed as a last step of our procedure
Rule-Based Procedure

• Approach is algebraic
  – Rules model interaction of string + arithmetic solvers
    • A closed derivation tree $\Rightarrow$ problem is UNSAT
    • A state where no rule applies $\Rightarrow$ 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

<table>
<thead>
<tr>
<th>Result</th>
<th>CVC4</th>
<th>Z3-STR</th>
<th>Kaluza</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsat</td>
<td>11,625&lt;sup&gt;1&lt;/sup&gt;</td>
<td>317</td>
<td>11,769&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>sat</td>
<td>33,271</td>
<td>1,583</td>
<td>31,372</td>
</tr>
<tr>
<td>unknown</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>timeout</td>
<td>2,388</td>
<td>2,123</td>
<td>84</td>
</tr>
<tr>
<td>error</td>
<td>0</td>
<td>120&lt;sup&gt;5&lt;/sup&gt;</td>
<td>1,140</td>
</tr>
</tbody>
</table>

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

- CVC4
- Z3-str
- Kaluza*
Further Work

• Theoretical:
  – Identify fragments when approach is refutation complete
    • [Abdullah et al CAV14]

• Regular language membership $\tau \in R^*$
  – 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: $x \cdot a = b \cdot x$
Challenge: Looping Word Equations

\[ x \cdot a = b \cdot x \]
\[ x = b \cdot x' \]

Variant of Original Equation!
Challenge: Looping Word Equations

\[ x \cdot a = b \cdot x \]

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

\[ x \cdot a = b \cdot x \iff \exists yz. (a = y \cdot z \land b = z \cdot y \land x \in (z \cdot y)^* z) \]