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

Tianyi Liang
Andrew Reynolds
Cesare Tinelli
Morgan Deters
Clark Barrett
Motivation: Security Applications

```c
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"))
```

Explain
Encode
Solve
Objectives

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

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

- Example input:
  \[
  \text{len}(x) > \text{len}(y) \\
  x \cdot z = y \cdot ab
  \]
Cooperating *Theory Solvers*

- \( \text{len}(x) > \text{len}(y) \)
- \( x \cdot z = y \cdot ab \)

- Distribute constraints to corresponding theory solvers

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

**Theory**
- Strings
  - \( x \cdot z = y \cdot ab \)
Cooperating *Theory Solvers*

- Communicate (dis)equalities over shared terms
  - [Nelson-Oppen]

- \( \text{len}(x) > \text{len}(y) \)
  - \( x \cdot z = y \cdot ab \)

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

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

• Uses procedure consisting of four steps:

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

- Add **equalities** to $A$ regarding the **length** of (non-variable) terms from $S$

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

**Theory**

- LIA

**A**

- $\text{len}(x) > \text{len}(y)$

**Theory**

- Strings

**S**

- $\text{len}(x) \neq \text{len}(y)$
- $x \cdot z = y \cdot ab$
Check Length Constraints

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

Theory LIA

$\Rightarrow$ Check if $A$ is satisfiable

Theory Strings

- $\text{len}(x) > \text{len}(y)$
- $\text{len}(x) + \text{len}(z) = \text{len}(y) + 2$
- $\text{len}(x) \neq \text{len}(y)$
- $x \cdot z = y \cdot ab$
Normalize Equalities

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

- 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 \( t \)

Strings

\[
\text{len}(x) \neq \text{len}(y) \\
x \cdot z = y \cdot ab
\]
Normalize Equalities

• Modified example:

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

• Flat form of terms from first equality are not the same:
  – \( F[z \cdot w] \) is: \( x \cdot a \cdot w \)
  – \( F[y \cdot ab] \) is: \( y \cdot ab \)

• Procedure continues based on three cases:
  – We know the length of \( x \) and \( y \) are equal: \textbf{conclude } x=y
  – We know the length of \( x \) and \( y \) are disequal: conclude \( \exists k.( ( x=y \cdot k \lor y=x \cdot k ) \land \text{len}(k)>0 ) \)
  – We know \textbf{neither} : guess their lengths are equal, restart

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

- After concluding \( x = y \),

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

- Flat form of terms from first equality are now, e.g.:
  - \( F[z \cdot w] \) is: \( y \cdot a \cdot w \)
  - \( F[y \cdot ab] \) is: \( y \cdot ab \)

- Will conclude \( w = b \), after which \( F[z \cdot w] = F[y \cdot ab] \)
Normalize Equalities

• 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

- For **disequalities** in $S$
  - A disequality $t \not= s$ is normalized if:
    - $\text{len}(t) \neq \text{len}(s)$, or
    - $N[t] = t_1 \cdot u \cdot t_2$ and $N[s] = s_1 \cdot v \cdot s_2$, where:
      - $\text{len}(t_1) = \text{len}(t_2)$,
      - $\text{len}(u) = \text{len}(v)$, and
      - $u \neq v$

- For example:
  - $\text{len}(z) \neq \text{len}(y)$
    - $z \neq y$
    - $x \cdot a \cdot z \neq x \cdot b \cdot z$
    - $x \cdot w \neq y \cdot b$
Normalize Disequalities

• 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

\[
\begin{align*}
\text{len}(z) &\neq \text{len}(y) \\
z &\neq y \\
x \cdot a \cdot z &\neq x \cdot b \cdot z \\
x \cdot w &\neq y \cdot b
\end{align*}
\]
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
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 \( \text{len}(x) \) and \( \text{len}(b) \), i.e. 1
  – If \( \text{len}(x) = 1 \), then \( x = a \) and \( x = b \) \( \Rightarrow \) conflict
  – If \( \text{len}(x) \neq 1 \)
    • 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:

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

<table>
<thead>
<tr>
<th></th>
<th>CVC4</th>
<th>Z3-STR</th>
<th>Kaluza</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>Incorrect</td>
<td>Incorrect</td>
<td>Incorrect</td>
</tr>
<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></td>
</tr>
<tr>
<td>error</td>
<td>0</td>
<td>120&lt;sup&gt;5&lt;/sup&gt;</td>
<td></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
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 $t \in \mathbb{R}^*$
  – 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:
    – `contains(x, y)`
    – `\neg 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/