Solving String Constraints via Hardware/Software Model Checking

Jie-Hong Roland Jiang†, Fang Yu §

†National Taiwan University
§National Chengchi University

Meeting on String Constraints and Applications (MOSCA'19),
May 6-9, 2019, Bertinoro, Italy
Outline

- Introduction
- Circuit based solving of string constraints (SLOG)
- Circuit based solving of string+length constraints (SLENT)
- Conclusions
Outline

- Introduction
  - Circuit based solving of string constraints (SLOG)
  - Circuit based solving of string+length constraints (SLENT)
- Conclusions
Motivating Example

Vulnerable to attack pattern: ' OR '1'='1'--

```php
<?PHP
$user_pass = $_POST["pwd"];
$user_id = $_POST["uid"];
$query = mysql_query("SELECT userName FROM accounts WHERE passwd = " . $user_pass . " AND userId = " . $user_id);
?>
```
Motivating Example

SELECT username FROM accounts WHERE passwd='' OR '1'='1'--

AND userId = '123'

SELECT username FROM accounts WHERE passwd='' OR '1'='1'--

AND userId = '123'

SELECT username FROM accounts WHERE passwd='' OR '1'='1'--

AND userId = '123'

SELECT username FROM accounts WHERE passwd='' OR '1'='1'--

AND userId = '123'
Introduction

String analysis

- Problem formulation
  - Input: an acyclic dependency graph extracted from string manipulating program
  - Output: possible string values for each string variable under program execution
Related Work

- Automata-based approaches
  - Use finite automata to characterize the set of possible string values in program execution
  - Can generate filter
  - Not support counterexample generation

- SMT-based approaches
  - Use string theory to decide whether a set of constraint is satisfiable
  - Can generate counterexample
  - Not support filter generation
Our Contribution

- We propose
  - Automata-based string analysis method using logic circuit representation
    - Advantages of logic circuit representation:
      - Support both counterexample generation and filter generation
      - Maintain circuit size linear in the input circuit sizes during string operations
Motivation of Circuit Representation

- Historic evolution of data structures and tools in logic synthesis and verification

![Diagram showing the evolution of circuit representation tools over time. The x-axis represents time from 1950 to 2000, and the y-axis represents problem size from 16 to 100000. The tools and technologies are labeled as follows:

- **1950-1970**: TT
- **1980**: SOP
- **1990**: BDD
- **2000**: AIG, CNF

Additional tools and technologies include:
- Espresso, MIS, SIS
- SIS, VIS, MVSIS
- ABC

Courtesy of Alan Mishchenko

2019/5/9
Introduction

Automata operation

\[ A_{lit_1} = \text{concat} \]

\[ A_1 = \text{CAT}(A_{lit_1}, A_{var_1}) \]

\[ A_2 = \text{CAT}(A_1, A_{lit_2}) \]

\[ A_3 = \text{CAT}(A_2, A_{var_2}) \]

\[ A_{sink} = A_3 \]
Introduction

Counterexample gen.

Goal: get witness of input string for vulnerable code

SELECT username FROM accounts WHERE passwd = '"

SELECT userName FROM accounts WHERE passwd=''

SELECT userId = 123

SELECT username FROM accounts WHERE passwd = 'OR '1'='1'--'

SELECT userId = 123

SELECT userName FROM accounts WHERE passwd = '"

SELECT userId = 123

SELECT userId = 123
Introduction

Filter generation

Goal: get black list for each var_node to filter out malicious input
Outline

- Introduction
- Circuit based solving of string constraints
- Circuit based solving of string+length constraints
- Conclusions
Preliminaries

A (nondeterministic) finite automaton $A = (Q, \Sigma, I, T, \mathcal{O})$
- $Q$: finite state set
- $\Sigma$: finite alphabet
- $I \subseteq Q$: set of initial states
- $T \subseteq \Sigma \times Q \times Q$: transition relation
- $\mathcal{O} \subseteq Q$: set of accepting states

Reserve a symbol for $\epsilon$ in $\Sigma$

$\mathcal{L}(A)$ denote the language (set of strings) accepted by $A$
Preliminaries

- Circuit representation
  - Boolean encoding on $Q$, $\Sigma$
  - Use characteristic functions $I(\hat{s})$, $T(\tilde{x}, \hat{s}, \hat{s}')$, $O(\hat{s})$ to represent an automaton

\[ \Sigma = \{ \epsilon, a \} \]
String/Automata Operations

- Supported operations
  - Intersection
  - Union
  - Concatenation
  - Replacement
  - Reverse
  - Prefix
  - Suffix
  - Emptiness checking
Intersection

- Circuit construction
  - Construct automaton \( A = INT(A_1, A_2) \) with
    \[ \mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2) : \]
    \[
    \begin{align*}
    T_{INT}(\vec{x}, \vec{s}, \vec{s}') &= T_1^\varepsilon(\vec{x}, \vec{s}_1, \vec{s}_1') \land T_2^\varepsilon(\vec{x}, \vec{s}_2, \vec{s}_2') \\
    I_{INT}(\vec{s}) &= I_1(\vec{s}_1) \land I_2(\vec{s}_2) \\
    O_{INT}(\vec{s}) &= O_1(\vec{s}_1) \land O_2(\vec{s}_2) \\
    \vec{s} &= (\vec{s}_1, \vec{s}_2)
    \end{align*}
    \]
Intersection

- Circuit construction

![Circuit Diagram]

\( T_{\text{INT}} \)
\( I_{\text{INT}} \)
\( O_{\text{INT}} \)
Intersection

Counterexample generation

\[
\begin{align*}
A_1: \ (p_1, \sigma_1, \ldots, \ p_\ell) \\
A_2: \ (q_1, \sigma_1, \ldots, \ q_\ell) \\
A: \ ((p_1, q_1), \sigma_1, \ (p_2, q_2), \sigma_2, \ldots, \ (p_\ell, q_\ell))
\end{align*}
\]

\text{INTCEx}
Intersection

- Filter generation
  - Let $B$ be the filter for $A = \text{Int}(A_1, A_2)$
  - $B$ can directly applied as a filter for $A_1$ as well as $A_2$
Circuit construction

Construct automaton \( A = UNI(A_1, A_2) \) with
\( \mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2) \):

\[
T_{UNI}(\tilde{x}, \tilde{s}, \tilde{s}') = \left( \neg \alpha \land \neg \alpha' \land T_1(\tilde{x}, \langle \tilde{s}_2 \rangle_m, \langle \tilde{s}_2' \rangle_m) \right) \lor \\
\left( \alpha \land \alpha' \land T_2(\tilde{x}, \tilde{s}_2, \tilde{s}_2') \right)
\]

\[
I_{UNI}(\tilde{s}) = \left( \neg \alpha \land I_1(\langle \tilde{s}_2 \rangle_m) \right) \lor \left( \alpha \land I_2(\tilde{s}_2) \right)
\]

\[
O_{UNI}(\tilde{s}) = \left( \neg \alpha \land O_1(\langle \tilde{s}_1 \rangle_m) \right) \land \left( \alpha \land O_2(\tilde{s}_2) \right)
\]

\( \tilde{s} = (\tilde{s}_2, \alpha) \)

\( \langle \tilde{s}_2 \rangle_m \): taking first \( m \) variables of \( \tilde{s}_2 \)
\( \alpha = 0 \): state in \( A_1 \), \( \alpha = 1 \): state in \( A_2 \)
Union

- Circuit construction
Union

- Counterexample generation

\[
\begin{align*}
A_1 &: (q_1, \sigma_1, q_2, \sigma_2, \ldots, q_\ell) & A_2 &: (\perp) \\
A &: ((q_1, c), \sigma_1, (q_2, c), \sigma_2, \ldots, (q_\ell, c)) \\
\text{UNICEX, } c = 0
\end{align*}
\]

\[
\begin{align*}
A_1 &: (\perp) & A_2 &: (q_1, \sigma_1, q_2, \sigma_2, \ldots, q_\ell) \\
A &: ((q_1, c), \sigma_1, (q_2, c), \sigma_2, \ldots, (q_\ell, c)) \\
\text{UNICEX, } c = 1
\end{align*}
\]
Filter generation

- Let $B$ be the filter for $A = Uni(A_1, A_2)$
- $B_1 = Int(B, A_1)$ and $B_2 = Int(B, A_2)$ form legitimate filter for $A_1$ and $A_2$, respectively
Concatenation

- Circuit construction
  - Construct automaton $A = CAT(A_1, A_2)$ with $\mathcal{L}(A) = \mathcal{L}(A_1) \cdot \mathcal{L}(A_2)$:

$$T_{CAT}(\vec{x}, \vec{s}, \vec{s}') = (\neg \alpha \land \neg \alpha' \land T_1(\vec{x}, \langle \vec{s}_2 \rangle_m, \langle \vec{s}'_2 \rangle_m)) \lor (\alpha \land \alpha' \land T_2(\vec{x}, \vec{s}_2, \vec{s}'_2)) \lor ((\vec{x} = \varepsilon) \land \neg \alpha \land \alpha' \land O_1(\langle \vec{s}_2 \rangle_m) \land I_2(\vec{s}'_2))$$

$$I_{CAT}(\vec{s}) = (\neg \alpha \land I_1(\langle \vec{s}_2 \rangle_m))$$

$$O_{CAT}(\vec{s}) = (\alpha \land O_2(\vec{s}_2))$$

$\vec{s} = (\vec{s}_2, \alpha)$

$\langle \vec{s}_2 \rangle_m$ : taking first $m$ variables of $\vec{s}_2$

$\alpha = 0$: state in $A_1$, $\alpha = 1$: state in $A_2$
Concatenation

- Circuit construction
Concatenation

- Counterexample generation

\[
A_1: (q_1, \sigma_1, \ldots, q_i) \quad \quad A_2: (q_{i+1}, \sigma_{i+1}, \ldots, q_n) \\
A_{\text{CAT}}: ((q_1, 0), \sigma_1 \ldots, (q_i, 0), \epsilon, (q_{i+1}, 1), \sigma_{i+1}, \ldots, (q_\ell, 1)) \quad \quad \text{CATCEX}
\]
Concatenation

Filter generation

- Let $B$ be the filter for $A = \text{Cat}(A_1, A_2)$
- First construct $B^\dagger = \text{Int}(A, B)$
- Let $B_1$ be a copy of $B^\dagger$ but with all the transition between states of $\alpha = 1$ being replaced with $\epsilon$-transition
- Let $B_2$ be a copy of $B^\dagger$ but with all the transition between states of $\alpha = 0$ being replaced with $\epsilon$-transition
Emptiness Checking

- Decide whether an automaton $A$ accept any string or not
- $T$ characterize one step of transition
- Convert into sequential circuit, then apply property directed reachability (PDR)
  - "pdr" command in Berkeley ABC only takes sequential circuits with *single initial state* and *transition functions*
Emptiness Checking

- Convert transition relation to transition functions
  - $n$ new variables $\tilde{y}$ for $n = |\tilde{s}|$
  - One new state variable $z$ with initial value 1
  - Output function: $\omega = (O(s) \land z)$
  - Next-state functions: $\delta_i = (y_i)$ for state variables $s_i$, $i = 1, \ldots, n$, and $\delta_{n+1} = (T(\tilde{x}, \tilde{s}, \tilde{y}) \land z)$ for state variable $z$
Emptiness checking

- Circuit construction

![Circuit Diagram]
Experiment Setting

- Compared string constraint solvers
  - Our tool: SLOG
  - Automata-based tools: JSA, Stranger
  - SMT-based tools: CVC4, Norn, Z3-str2

- Environment
  - Intel Xeon 8-core CPU
  - 16GB memory
  - Ubuntu 12.04 LCS
Experimental Setting

- 20000+ string analysis instances generated from web applications
  - Replacement-free small instances
  - Replacement-free large instances
  - Instances with replacement operation
Experimental Results (1/4)

- Replacement-free small instances

![Graph showing the performance of different tools across instances]

- SLOG
- Stranger
- CVC4
- JSA
- Z3-STR2
- Norn

Instances:
- 13943
- 16542
- 18829
- 20091

2019/5/9 MOSCA 2019
Experimental Results (2/4)

- Replacement-free large instances

![Graph showing performance of different tools over instances over time.](image-url)
Experimental Results (3/4)

- Instances with replacement operation
Experimental Results (4/4)

- SLOG performance on counterexample generation

<table>
<thead>
<tr>
<th>Group</th>
<th>#SAT</th>
<th>SolveTime (s)</th>
<th>CexGenTime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>8426</td>
<td>60664</td>
<td>481</td>
</tr>
<tr>
<td>Large</td>
<td>22</td>
<td>4236</td>
<td>18</td>
</tr>
<tr>
<td>Replacement</td>
<td>236</td>
<td>1015</td>
<td>25</td>
</tr>
</tbody>
</table>

accumulated solving and counterexample generation time
Outline

- Introduction
- Circuit based solving of string constraints (SLOG)
- Circuit based solving of string+length constraints (SLENT)
- Conclusions
Length Tracking

- Circuit construction
  - Construct automaton $A^{L} = TrkLen(A)$ with:

$$T^{L}(\tilde{x}, \tilde{s}, n, s', n) = T(\tilde{x}, \tilde{s}, s') \land$$
$$((\tilde{x} \neq \epsilon) \land (n' = n + 1)) \lor$$
$$((\tilde{x} = \epsilon) \land (n' = n)))$$

$$I^{L}(\tilde{s}) = I(\tilde{s})$$
$$O^{L}(\tilde{s}) = O(\tilde{s})$$
Intersection

- Circuit construction
  - Construct automaton $A = INT(A_1, A_2)$ with $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$:

$$T_{INT} \left( \vec{x}, \vec{s}, \vec{n}, \vec{s}', \vec{n}' \right) = T_1^e \left( \vec{x}, \vec{s}, \vec{n}, \vec{s}', \vec{n}' \right) \land T_2^e \left( \vec{x}, \vec{s}, \vec{n}, \vec{s}', \vec{n}' \right)$$

$$I_{INT} (\vec{s}) = I_1 (\vec{s}_1) \land I_2 (\vec{s}_2)$$

$$O_{INT} (\vec{s}) = O_1 (\vec{s}_1) \land O_2 (\vec{s}_2)$$

$$\vec{s} = (\vec{s}_1, \vec{s}_2)$$
Intersection

- Circuit construction

\[ T_{INT} \]

\[ T^c_1 \quad T^c_2 \]

\[ s \quad x \quad n \quad s' \quad n' \]

\[ I_{INT} \]

\[ I_1 \quad I_2 \]

\[ s_1 \quad s_2 \]

\[ O_{INT} \]

\[ O_1 \quad O_2 \]

\[ s_1 \quad s_2 \]
Circuit construction

Construct automaton \( A = UNI(A_1, A_2) \) with \( \mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2) \):

\[
T_{UNI} \left( \tilde{x}, \tilde{s}, \tilde{n}, \tilde{s}', \tilde{n}' \right) = \left( \neg \alpha \land \neg \alpha' \land T_1 \left( \tilde{x}, \langle \tilde{s}_2 \rangle_m, \tilde{n}, \langle \tilde{s}_2 \rangle_m, \tilde{n}' \right) \right) \lor \\
\left( \alpha \land \alpha' \land T_2 \left( \tilde{x}, \tilde{s}_2, \tilde{n}, \tilde{s}_2', \tilde{n}' \right) \right)
\]

\[
I_{UNI}(\tilde{s}) = \left( \neg \alpha \land I_1(\langle \tilde{s}_2 \rangle_m) \right) \lor \left( \alpha \land I_2(\tilde{s}_2) \right)
\]

\[
O_{UNI}(\tilde{s}) = \left( \neg \alpha \land O_1(\langle \tilde{s}_1 \rangle_m) \right) \land \left( \alpha \land O_2(\tilde{s}_2) \right)
\]

\( \tilde{s} = (\tilde{s}_2, \alpha) \)

\( \langle \tilde{s}_2 \rangle_m \): taking first \( m \) variables of \( \tilde{s}_2 \)

\( \alpha = 0 \): state in \( A_1 \), \( \alpha = 1 \): state in \( A_2 \)
Union

- Circuit construction
Concatenation

- Circuit construction

  - Construct automaton \( A = CAT(A_1, A_2) \) with
    \[ \mathcal{L}(A) = \mathcal{L}(A_1). \mathcal{L}(A_2) : \]

    \[
    T_{CAT} \left( \tilde{x}, \tilde{s}, \tilde{n}, \tilde{s}', \tilde{n}' \right) = \left( \neg \alpha \land \neg \alpha' \land T_1(\tilde{x}, (\tilde{s}_2)_m, \tilde{n}_1, (\tilde{s}_2')_m, \tilde{n}'_1) \land (\tilde{n}_2 = \tilde{n}'_2) \right) \lor
    \left( \alpha \land \alpha' \land T_2(\tilde{x}, \tilde{s}_2, \tilde{n}_2, \tilde{s}'_2, \tilde{n}'_2) \land (\tilde{n}_1 = \tilde{n}'_1) \right) \lor
    \left( (\tilde{x} = \epsilon) \land \neg \alpha \land \alpha' \land O_1((\tilde{s}_2)_m) \land I_2(\tilde{s}'_2) \land (\tilde{n} = \tilde{n}') \right)
    \]

    \[
    I_{CAT}(\tilde{s}) = \left( \neg \alpha \land I_1((\tilde{s}_2)_m) \right)
    \]

    \[
    O_{CAT}(\tilde{s}) = \left( \alpha \land O_2(\tilde{s}_2) \right)
    \]

    \[
    \tilde{s} = (\tilde{s}_2, \alpha)
    \]

    \[
    (\tilde{s}_2)_m : \text{taking first } m \text{ variables of } \tilde{s}_2
    \]

    \[
    \alpha = 0: \text{state in } A_1, \alpha = 1: \text{state in } A_2
    \]
Concatenation

- Circuit construction

\[ T_{\text{CAT}} \]

\[ n_2 = n_2' \]
\[ \alpha \quad \alpha' \]
\[ n = n' \]
\[ n_1 = n_1' \]

\[ T_1 \]

\[ x \quad s_2 \quad n \quad s_2' \quad n' \]

\[ T_2 \]

\[ I_{\text{CAT}} \]

\[ \langle s_2 \rangle_m \]

\[ O_{\text{CAT}} \]

\[ s_2 \]
Prefix

- Circuit construction

Diagram showing circuit construction with labels and symbols.
Experiment Setting

- Compared string constraint solvers
  - Our tool: SLENT (using IC3_{IA} model checker)
  - Automata-based tools: (UCSB) ABC
  - SMT-based tools: CVC4, Norn, S3P, Trau, Z3-str3

- Environment
  - Intel Xeon 8-core CPU
  - 16GB memory
  - Ubuntu 12.04 LCS
Experimental Results (1/3)

- 2000+ instances converted from Kaluza benchmarks involving only string concatenation and length constraints

<table>
<thead>
<tr>
<th>solver</th>
<th>time (s)</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>#TO (200s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z3str3</td>
<td>56.46</td>
<td>1017</td>
<td>983</td>
<td>0</td>
</tr>
<tr>
<td>CVC4</td>
<td>88.89</td>
<td>1017</td>
<td>983</td>
<td>0</td>
</tr>
<tr>
<td>Norn</td>
<td>2025.30</td>
<td>1013</td>
<td>983</td>
<td>4</td>
</tr>
<tr>
<td>ABC</td>
<td>255.76</td>
<td>1013</td>
<td>983</td>
<td>4</td>
</tr>
<tr>
<td>S3P</td>
<td>137.90</td>
<td>1015</td>
<td>983</td>
<td>2</td>
</tr>
<tr>
<td>Trau</td>
<td>123.85</td>
<td>1017</td>
<td>983</td>
<td>0</td>
</tr>
<tr>
<td>SLENT</td>
<td>1397.82</td>
<td>1013</td>
<td>983</td>
<td>4</td>
</tr>
</tbody>
</table>
Experimental Results (2/3)

- 236 instances converted from Stranger benchmarks with involving string-to-string replaceall, concatenation, and length constraints

<table>
<thead>
<tr>
<th>solver</th>
<th>time (s)</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>#TO (600s)</th>
<th>#abort</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>2282.84</td>
<td>109(31)</td>
<td>111(0)</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>S3P</td>
<td>605.79</td>
<td>30(0)</td>
<td>114(3)</td>
<td>22</td>
<td>70</td>
</tr>
<tr>
<td>Trau</td>
<td>687.49</td>
<td>54(2)</td>
<td>139(22)</td>
<td>5</td>
<td>38</td>
</tr>
<tr>
<td>SLENT</td>
<td>26692.55</td>
<td>88(0)</td>
<td>141(0)</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
Experimental Results (3/3)

- 101 instances converted from Stranger benchmarks with involving language-to-language replaceall, concatenation, and length constraints

<table>
<thead>
<tr>
<th>solver</th>
<th>time (s)</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>#TO (600s)</th>
<th>#abort</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>977.80</td>
<td>46 (2)</td>
<td>41 (0)</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>SLENT</td>
<td>4413.25</td>
<td>44 (0)</td>
<td>38 (0)</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

9 under TO 1800s
Conclusions

- SLOG
  - Logic circuit based automata manipulation method for string analysis
    - Support both counterexample generation and filter synthesis
    - Maintain circuit size linear in input circuit sizes for string operations
    - Postpone language emptiness checking with model checking at the end
Conclusions (cont’d)

- **SLENT**
  - Encode length information to string automata as length-encoded automata
    - Construct characteristic functions of length-encoded automata through automata manipulations that correspond to string and length constraints
    - Leverage a symbolic model checker for infinite state systems as an engine for language emptiness checking
Conclusions (cont’d)

- SLOG and SLENT are based on scalable circuit representation and good at solving complex string constraints
- Counterexample generation and filter synthesis are possible
- Future work
  - Support complement operation
  - Allow relation on string variables
Acknowledgement

Shih-Yu Chen  Chun-Han Lin  Tsung-Lin Tsai  Hung-En Wang
Thanks for Your Attention!

- Questions?