(Un)-Decidability Results for Word Equations with Length and Regular Expression Constraints

Vijay Ganesh

November 22, 2013
Web Security: A Motivation for Theories over Strings, Length and RE

Motivating problem:

- Web apps are plagued by security errors
- Examples include SQL injection, XSS, and CSRF attacks
- These security errors often involve some computation over strings, numbers and regular expressions

Question:

- How can we analyze code automatically to detect/find these class of errors?

Solution:

- String analysis (static, dynamic, symbolic) that uses a backend string solver
String Solver Usage Scenario:
For Security Analysis, Verification and Bug-finding

Program Reasoning Tool

String Program

Specification

String Formulas

String Solver

Program is Correct?

or Generate Tests

SAT/UNSAT
Web Security: 
A Motivation for Theories over Strings and Numbers

How are strings, length and RE used in Web applications?

- Construct SQL commands
  - string concatenations to create strings from input forms
- Sanity check
  - membership in regexp for checking input against known vectors
- Length check
  - string length comparisons for protecting against overflow errors

Powerful language over strings & numbers

- String operations such as concatenation, extraction
- Length function: strings -> int
- Regular expressions and context-free grammars
String Solver Usage Scenario:
For Security Analysis, Verification and Bug-finding

String Program  Specification

Program Reasoning Tool

String Formulas

Program is Correct?
or Generate Tests

String Solver

SAT/UNSAT
Web Security: A Motivation for Theories over Strings and Numbers

SAT procedures for theories of strings, length and regular expressions

- **HAMPI** [G. et al. (ISSTA 2009, CAV 2011, TOSEM 2012)]
  - Supports bounded strings, bounded regular expressions and context-free grammars (finite languages)

- **Rex** [Veanes et al.]
  - Unbounded strings and length

- **Kaluza** [Saxena et al.]
  - Bounded strings and length

- **Z3-str** [Zheng, Zhang and G. (FSE 2013)]
  - Unbounded strings and length
A Rich Language for Strings (Words), Length and RE:
For Security, Verification and Bug-finding Applications

<table>
<thead>
<tr>
<th>string sort</th>
<th>number sort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constants</strong></td>
<td>finite strings/words over finite alphabet $\Sigma$ $\epsilon, a, b, ab, \ldots \in \Sigma^*$</td>
</tr>
<tr>
<td><strong>Variables</strong></td>
<td>denoted by $X,Y,\ldots$</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>string concatenation $\cdot$</td>
</tr>
<tr>
<td><strong>Predicates</strong></td>
<td>word equations $\text{str-term} = \text{str-term}$</td>
</tr>
<tr>
<td><strong>Predicates</strong></td>
<td>regexp membership $\text{str-term} \in \text{RegExp}$</td>
</tr>
</tbody>
</table>

Formulas constructed inductively in the usual way
Sample formulas

- $abX = Xba$, where $X$ is a string variable
- $(abX = Xba) \land (X \in (ab \mid ba)(ab)^*a) \land (\text{len}(X) \leq 5))$

Informal semantics

- A solution to a word equation is a mapping from variables to string constants
- A word equation may have infinitely many solutions
- For example: the solutions to the equation $Xa = aX$ can be represented by the set $a^*$
- Example of a formula with RE: $aX \in (a + b)^*$
- Observe that word equations can represent regular sets or even CFGs
More sample formulas

- \( abX = Xba \), where \( X \) is a string variable
  - Solution: \( X := a \)

- \((abX = Xba) \land (X \in (ab | ba)(ab)^*a) \land (\text{len}(X) \leq 5))\)
  - Solution: \( X := aba \)

- \( Xa = bX \)
  - UNSAT

- \( XabX = XbaX \)
  - UNSAT

- Difficulty: Overlapping variables
Outline of the Rest of the Talk

- Undecidability result #1
  - $\forall \exists$-fragment of positive word equations is undecidable

- Decidability result #2
  - The SAT problem for word equations in solved form + length constraints is decidable

- Decidability result #3
  - The SAT problem for word equations in solved form + length + RE constraints is decidable

- Practical utility
  - Most word equations in applications are already in solved form

- Solvers: HAMPI and Z3-str

- Future directions
Boundary between decidability and undecidability for word equations

- Fully quantified theory with only word equations is undecidable (Quine 1946)
- Quantifier-free (QF) theory of word equations is decidable (Makanin 1977)
- QF fragment with only word equations is PSPACE (Plandowski 2006)
- How many quantifier-alternation for theory to be undecidable? (Our Answer: 1)
Decidability of SAT problem of QF fragment with word equations and integer constraints over length function

- Open problem for 70 years  
  (Studied by Post, Matiyasevich and Plandowski)

- If shown undecidable, lead to new proof of Matiyasevich’s theorem  
  (Matiyasevich 2006)

- Matiyasevich showed that Hilbert’s Tenth problem is unsolvable  
  (Matiyasevich 1971)

- We provide a conditional solution important in practice
Decision problem #3: SAT Problem for QF Word Equations, Length, Regexp

Decidability of SAT problem of QF fragment with word equations, length function and regular expression membership predicate

- Still open
- Word equations, membership predicate over regexp is decidable (Schulz 1992. Extending Makanin’s 1977 result)
- Restriction: \( \text{var} \in \text{regexp} \)
- Strictly more general than Makanin’s result
- We provide a conditional solution important in practice
Result #1:
Undecidability of $\forall \exists$-fragment over Word Equations

Theorem:
Validity problem for $\forall \exists$-sentences over positive word equations, with at most two occurrences of any variable is undecidable

Proof Idea:
- Reduction from the halting problem for two-counter machines to the problem-under-consideration
- Two-counter machines can simulate arbitrary Turing machines
- Hence, halting problem for two-counter machines is undecidable
- Choice of two-counter machines is crucial for our proof
Undecidability Result:
Recalling Two-counter Machines

Finite State Control
of Two-counter Machine

Read-only Input Tape

R/W Storage Tape 1
Z B ...
B ...

R/W Storage Tape 2
Z C ...
C ...

q_j in Q
Undecidability Result: Instantaneous Description (ID) and Strings

Finite State Control of Two-counter Machine

Read-only Input Tape

\[ ... \quad W_i \quad ... \]

R/W Storage Tape 1

\[ Z \quad B \quad ... \quad B \quad ... \]

R/W Storage Tape 2

\[ Z \quad C \quad ... \quad C \quad ... \]

\[ q_j \text{ in } Q \]
Undecidability Result:
Initial Instantaneous Description (ID)

Initial ID = \(<q_0, W_0, \epsilon, \epsilon>\)
Undecidability Result:
Final Instantaneous Descriptions (ID)

Final ID = \(<q_f, W_0, \epsilon, \epsilon>\)
Given a machine M and an input string W, define well-formed computational history as a string:

\[ #q_0W0 \in \varepsilon \# \ldots \#D_i \#ID_{i+1} \# \ldots \]

Accepting computational history:

\[ #q_0W0 \in \varepsilon \# \ldots \#ID_i \#ID_{i+1} \# \ldots \#q_fW0 \in \varepsilon \]

Computational history can be accepting, rejecting, non well-formed or non-terminating
Undecidability Result: 
Revisiting the Proof Strategy

By reduction from halting problem of two-counter machines to the validity problem:

given a machine M and an input string w, construct a string formula $\theta$ such that

- Every assignment to the $\theta$ is a non-accepting computational history
- M does not halt on w $\iff$ $\theta$ is valid
- Intuition: M does not halt on w iff no finite computational history is an accepting history

String alphabet over which $\theta$ is defined

- $\Sigma_0 := \#q_i w N_j : q_i \in Q, 0 \leq j < |w|$
- $\Sigma_1 := b$
- $\Sigma_2 := c$
Undecidability Result:
How does the Reduction Work? The structure of $\theta$

given a machine $M$ and an input string $w$, the string formula $\theta$
satisfied by any non-accepting computational history:

For any string $S$, there exists partitions such that:
$(\forall S \exists \text{parts} . \theta(S, \text{parts}) \text{ is valid } \iff (M, w) \text{ does not accept and halt})$

- Does not begin with an Initial ID
- OR
- Does not end in a Final ID
- OR
- $ID_{i+1}$ does not follow $ID_i$ according to transition function of $M$
- OR
- Is not a well-formed sequence of IDs
Undecidability Result:
How does the Reduction Work? The structure of $\theta$

$\theta$ accepts assignments that are not well-formed sequences of IDs

This sub-formula illustrates the value of two-counter machines over Turing

- Well-formed ID $\in \text{regexp } \Sigma_0b^*c^*$
- Well-formed sequence of IDs $\in \text{regexp } (\Sigma_0b^*c^*)^*$
- Not a well-formed sequence of IDs $\in \Sigma^* - (\Sigma_0b^*c^*)^*$
- Regexp can be eliminated by word equations:
  $$(S = \epsilon) \lor (S = S_1 \cdot c \cdot b \cdot S_4)$$

Lesson confirmation: good to have multiple representations and proofs!
Recap of Result #1: Undecidability of $\forall\exists$-fragment over Word Equations

Theorem:
Validity problem for $\forall\exists$-sentences over positive word equations, with at most two occurrences of any variable is undecidable.

Proof Idea:
- Reduction from the halting problem for two-counter machines to the problem-in-question.
- The halting problem for two-counter machines is known to be undecidable.
- Encode computation histories as solutions to word equations.
- Choice of two-counter machines is crucial.
- Given $M, w$, construct $\theta$ such that $M$ does not halt and accept $w \iff \theta$ valid.
Result #2: The SAT Problem for Word Equations and Length

The Problem:

Is the SAT problem for the QF theory of word equations and length constraints decidable?

Example:

\[ abX = Xba \land X = abY \land \text{len}(X) < 2 \]

Importance:

- Problem studied for 70 years, still open
- Directly relevant to JavaScript bug-finding
- If proven undecidable, provides a new simpler proof for Matiyasevich’s theorem (Matiyasevich 2006)
Result #2: Difficulty of Resolving SAT Problem for Word Equations and Length

The Problem:
Is the SAT problem for the QF theory of word equations and length constraints decidable?

Difficulty:
- Word equations, by themselves, are decidable (Makanin)
- Length constraints are essentially Presburger arithmetic
- However, no known finite way to characterize length constraints implied by word equations
- No known theorem that states that implied length constraints cannot be finitized
Result #2:
Conditional Decidability of SAT Problem for Word Equations and Length

Theorem: This SAT problem is decidable, if word equations can be converted into solved form.

Solved form for conjunction of word equations:

- Every word equation can be written as $X = t$
- $t$ is a concatenation of string constants, int parameters and new variables
- Variable can occur exactly once, and only on the left handside

Example: $Xa = aY \land Ya = Xa$

- Solved form: $X = a^i$, $Y = a^i$ where $i \geq 0$
Results #2 and #3: Conditional SAT Procedure for Word Equations and Length

**Solved form:** finite representation of implied length constraints

**Result #3:** Easy extension when regular expressions are added

**Satisfiability procedure:** Suffices to consider conjunction of literals
Results #2 and #3: Relevance of Solved Form to Practice

**Kaluza**: A solver for word equations and length constraints (Saxena, Akhawe, Song)

- 50,000+ constraints from JavaScript bug-finding applications
- Categorized into SAT and UNSAT constraints
- Over 75% and 87% of the word equations in solved form
- Uses the HAMPI string solver (G. et al)
Related Work

Undecidability of free semi-groups
(Durnev 1995, Marchenkov 1982)

Differences:
- Our proof uses standard well-understood two-counter machines
- Durnev uses fewer variables, more occurrences
- Free semi-groups don’t have identity operator

Undecidability of bit-vectors with unbounded concatenation, extraction and equality
(Moller 1998)

Differences:
- Our result is stronger, because the theory we consider is weaker
HAMPI and Z3-str String Solvers

HAMPI String Solver

- \( X = \text{concat(“SELECT...”,v)} \land (X \in \text{SQL\_grammar}) \)
- JavaScript, PHP, ... string expressions
- NP-complete
- ACM Distinguished Paper Award 2009
- Google Faculty Research Award 2011
## Theory of Strings

### The Hampi Language

<table>
<thead>
<tr>
<th>PHP/JavaScript/C++...</th>
<th>HAMPI: Theory of Strings</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var a; $a = 'name'</td>
<td>Var a : 1...20; a = 'name'</td>
<td>Bounded String Variables String Constants</td>
</tr>
<tr>
<td>string_expr.&quot; is &quot;</td>
<td>concat(string_expr,&quot; is &quot;);</td>
<td>Concat Function</td>
</tr>
<tr>
<td>substr(string_expr,1,3)</td>
<td>string_expr[1:3]</td>
<td>Extract Function</td>
</tr>
<tr>
<td>assignments/strcmp</td>
<td>equality</td>
<td>Equality Predicate</td>
</tr>
<tr>
<td>a = string_expr; a /= string_expr;</td>
<td>a = string_expr; a /= string_expr;</td>
<td></td>
</tr>
<tr>
<td>Sanity check in regular expression RE</td>
<td>string_expr in RE</td>
<td>Membership Predicate</td>
</tr>
<tr>
<td>Sanity check in context-free grammar CFG</td>
<td>string_expr in SQL</td>
<td></td>
</tr>
<tr>
<td>string_expr NOT in SQL</td>
<td>string_expr NOT?contains sub_str</td>
<td></td>
</tr>
<tr>
<td>string_expr contains a sub_str</td>
<td>string_expr contains sub_str</td>
<td>Contains Predicate (Substring Predicate)</td>
</tr>
<tr>
<td>string_expr does not contain a sub_str</td>
<td>string_expr NOT?contains sub_str</td>
<td></td>
</tr>
</tbody>
</table>

Ganesh et al

Word Equations with Length Constraints

November 22, 2013
HAMPI and Z3-str String Solvers

HAMPI Solver Motivating Example

SQL Injection Vulnerabilities

SELECT m FROM messages WHERE id='1' OR 1 = 1
HAMIPI and Z3-str String Solvers

HAMIPI Solver Motivating Example

SQL Injection Vulnerabilities

Buggy Script

\[
\begin{align*}
\text{if } & (\text{input in regexp}([0-9]+)) \\
\text{query} & := \text{“SELECT m FROM messages WHERE id='" + input + "'"} 
\end{align*}
\]

- input passes validation (regular expression check)
- query is syntactically-valid SQL
- query can potentially contain an attack substring (e.g., 1' OR '1' = '1')
HAMPI and Z3-str String Solvers

HAMPI Solver Motivating Example
SQL Injection Vulnerabilities

should be: “^[0-9]+$”

Buggy Script

if (input in regexp(“[0-9]+”))
   query := “SELECT m FROM messages WHERE id=‘ ’ + input + ‘ ’”

• input passes validation (regular expression check)
• query is syntactically-valid SQL
• query can potentially contain an attack substring (e.g., 1’ OR ‘1’ = ‘1)
**HAMPI and Z3-str String Solvers**

**HAMPI Solver Motivating Example**

**SQL Injection Vulnerabilities**

```plaintext
if (input in regexp("[0-9]+"))
   query := "SELECT m FROM messages WHERE id='  ' + input + ' '"
```

---

**Diagram**

- **Program Reasoning Tool**
  - **Specification**
  - **String Formulas**
    - **HAMPI**
    - **SAT/UNSAT**
    - **Generate Tests/Report Vulnerability**
Expressing the Problem in HAMPI

SQL Injection Vulnerabilities

Input String

Var v : 12;

cfg SqlSmall := "SELECT " [a-z]+ " FROM " [a-z]+ " WHERE " Cond;

cfg Cond := Val"=" Val | Cond " OR " Cond;

cfg Val := [a-z]+ | "" [a-z0-9]* "" | [0-9]+;

SQL Query

val q := concat("SELECT msg FROM messages WHERE topicid=" , v , ");

assert v in [0-9]+;

assert q in SqlSmall;

assert q contains "OR '1'='1';";

“q is a valid SQL query”

“q contains an attack vector”
Expressing the Problem in HAMPI

SQL Injection Vulnerabilities

Input String

Var v : 12;

cfg SqlSmall := "SELECT " [a-z]+ " FROM " [a-z]+ " WHERE " Cond;

cfg Cond := Val"=" Val | Cond " OR " Cond;

cfg Val := [a-z]+ | "" [a-z0-9]* "" | [0-9]+;

SQL Query

val q := concat("SELECT msg FROM messages WHERE topicid="', v, "'");

assert v in [0-9]+;

assert q in SqlSmall;

assert q contains "OR '1'='1'";

“q is a valid SQL query”

“q contains an attack vector”
Hampi Key Conceptual Idea
Bounding, expressiveness and efficiency

<table>
<thead>
<tr>
<th>Li</th>
<th>Complexity of $\emptyset = L_1 \cap \ldots \cap L_n$</th>
<th>Current Solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context-free</td>
<td>Undecidable</td>
<td>n/a</td>
</tr>
<tr>
<td>Regular</td>
<td>PSPACE-complete</td>
<td>Quantified Boolean Logic</td>
</tr>
<tr>
<td>Bounded</td>
<td>NP-complete</td>
<td>SAT Efficient in practice</td>
</tr>
</tbody>
</table>
HAMPI and Z3-str String Solvers

Hampi Key Idea: Bounded Logics
Testing, Analysis, Vulnerability Detection,...

- Finding SAT assignment is key
- Short assignments are sufficient

- Bounding is sufficient
- Bounded logics easier to decide
HAMPI: Result 1
Static SQL Injection Analysis

- 1367 string constraints from Wasserman & Su [PLDI’07]
- Hampi scales to large grammars
- Hampi solved 99.7% of constraints in < 1 sec
- All solvable constraints had short solutions
HAMPI and Z3-str String Solvers

**HAMPI: Result 2**

Security Testing and XSS

- Attackers inject client-side script into web pages
- Somehow circumvent same-origin policy in websites
- `echo “Thank you $my_poster for using the message board”;`
- Unsanitized `$my_poster`
- Can be JavaScript
- Execution can be bad
HAMPI and Z3-str String Solvers

**HAMPI: Result 2**

Security Testing

• Hampi used to build Ardilla security tester [Kiezun et al., ICSE’09]

• 60 new vulnerabilities on 5 PHP applications (300+ kLOC)
  • 23 SQL injection
  • 37 cross-site scripting (XSS)

• 46% of constraints solved in < 1 second per constraint

• 100% of constraints solved in <10 seconds per constraint

5 added to US National Vulnerability DB
HAMPI and Z3-str String Solvers

HAMPI: Result 3
Comparison with Competing Tools

- HAMPI vs. CFGAnalyzer (U. Munich): HAMPI ~7x faster for strings of size 50+
- HAMPI vs. Rex (Microsoft Research): HAMPI ~100x faster for strings of size 100+
- HAMPI vs. DPRLE (U. Virginia): HAMPI ~1000x faster for strings of size 100+
**Z3-str String Solver**

- Quantifier-free theory of word equations and length function
- **Status:** unknown
- Our partial decidability technique
  - Given a word equation partition its solutions space into finite buckets
  - Leverage Z3 for identifying equivalent expressions and length consistency checks
  - Approximate by heuristically solving “overlapping” equations

*Joint work with Xiangyu Zhang and Yunhui Zheng (Purdue University)*
**Z3-str String Solver**

- Quantifier-free theory of word equations and length function
- Status: unknown
- Our partial decidability technique
  - Given a word equation partition its solutions space into finite buckets
  - Leverage Z3 for identifying equivalent expressions and length consistency checks
  - Approximate by heuristically solving “overlapping” equations

*Joint work with Xiangyu Zhang and Yunhui Zheng (Purdue University)*
### Key Contributions

https://ece.uwaterloo.ca/~vganesh

<table>
<thead>
<tr>
<th>Name</th>
<th>Key Concept</th>
<th>Impact</th>
<th>Pubs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STP</strong> Bit-vector &amp; Array Solver</td>
<td>Abstraction-refinement for Solving</td>
<td>Concolic Testing</td>
<td>CAV 2007</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CCS 2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TISSEC 2008</td>
</tr>
<tr>
<td><strong>HAMPI</strong> String Solver</td>
<td>App-driven Bounding for Solving</td>
<td>Analysis of Web Apps</td>
<td>ISSTA 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TOSEM 2012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CAV 2011</td>
</tr>
<tr>
<td><strong>(Un)Decidability</strong> results for Strings</td>
<td>Reduction from two-counter machine halting problem</td>
<td></td>
<td>HVC 2012</td>
</tr>
<tr>
<td><strong>Taint-based Fuzzing</strong></td>
<td>Information flow is cheaper than concolic</td>
<td>Scales better than concolic</td>
<td>ICSE 2009</td>
</tr>
<tr>
<td><strong>Automatic Input Rectification</strong></td>
<td>Acceptability Envelope: Fix the input, not the program</td>
<td>New way of approaching SE</td>
<td>ICSE 2012</td>
</tr>
</tbody>
</table>

1. **STP** won the SMTCOMP 2006 and 2010 competitions for bit-vector solvers
2. **HAMPI**: ACM Best Paper Award 2009
3. Google Award 2011
4. Retargetable Compiler (DATE 1999)
5. Proof-producing decision procedures (TACAS 2003)
6. Error-finding in ARBAC policies (CCS 2011)
7. Programmatic SAT Solvers (SAT 2012)
Summary of Results

0) Motivated by Web security
   - Considered powerful theory over word eqns, length, and regexp

1) Undecidability of $\forall \exists$ fragment over word equations
   - Interesting use of two-counter machines

2) Conditional decidability of SAT for QF word eqns and length
   - Relies on solved form
   - Empirically observed the value of solved form in practice

3) Extended result #2 to QF word eqns, length, regexp

4) HAMPI and Z3-str

5) Formal methods for counterexample construction