Overview

String Constraints → Model Counter → Number of Solutions
Can you solve it, Will Hunting?

Can you solve it, Will Hunting?
A Motivating Example

An adversary learns a password. User must select a new password.

Policy for selecting a new password.

```java
public Boolean NewPWCheck(String new_p, old_p){
    if( old_p.contains(new_p) || ...
        new_p.contains(old_p) || ...
        old_p.reverse().contains(new_p)) || ...
        new_p.contains(old_p.reverse()) ){
        System.out.println("Too similar.");
        return false;
    } else
    return true;
}
```
A Motivating Example

Suppose an adversary knows $\text{old}_p = "\text{abc-16}"$ and knows the policy.

<table>
<thead>
<tr>
<th>Constraints on possible values of $\text{NEW}_P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(not (contains (toLowerCase $\text{NEW}_P$) &quot;abc-16&quot;))</td>
</tr>
<tr>
<td>(not (contains (toLowerCase $\text{NEW}_P$) &quot;61-cba&quot;))</td>
</tr>
<tr>
<td>(not (contains &quot;abc-16&quot; (toLowerCase $\text{NEW}_P$)))</td>
</tr>
<tr>
<td>(not (contains &quot;61-cba&quot; (toLowerCase $\text{NEW}_P$)))</td>
</tr>
</tbody>
</table>

If password length $= n$, then there are $|\Sigma|^n$ possible passwords.

If adversary knows $\text{old}_p$ and the policy . . .

▶ how much is the reduction in search space?
▶ what is the probability of guessing the new password?

Motivation

In general, we want to answer questions regarding

▶ probability of program behaviors,
▶ number of inputs that cause an error,
▶ amount of information flow,
▶ information leakage,
▶ other, as yet unforeseen, applications...

These are quantitative questions which require model counting.
Motivation

Techniques for model counting for other theories

Boolean Logic Formulas
- DPLL
- Random sampling based
- Approximations

Linear Integer Arithmetic:
- LattE
- Barvinok

Motivation

String manipulating programs are pervasive
- security critical functions,
- server side sanitization functions,
- databases,
- dynamic code generation.

We need model counting for strings in order to make quantitative guarantees about these types of programs.

Software for string constraint model counting
- Automata-Based Model Counter (ABC) [Aydin, et. al. CAV 2015]
- String Model Counter (SMC) [Luu, et. al. PLDI 2014]
Motivation and Background

Model Counting Boolean Formulas

String Model Counting
  ▶ Automata-Based Methods
  ▶ Non-Automata-Based Method

String Model Counting Benchmarks

Model Counting

Recall the classic (boolean) SAT problem

Given a formula $\phi$ from propositional logic, is it possible to assign all variables the values $T$ (true) or $F$ (false) so that the formula is true?

Example:

$$\phi = (x \lor y) \land (\neg x \lor z) \land (z \lor w) \land x \land (y \lor v)$$

$\phi$ is satisfiable by setting

$$(x, y, z, w, v) = (T, F, T, F, T).$$

A satisfying assignment is called a model for $\phi$. 
Model Counting

The **model counting problem**

Given a formula $\phi$ over some theory (Boolean, LIA, Strings, ...)

**how many models are there** for $\phi$?

Difficulty of Model Counting

Model counting is “at least as hard” as satisfiability check.

$$|\phi| > 0 \iff \phi \text{ is satisfiable}$$

Model Counting Boolean SAT

$$\phi = (x \lor y) \land (\neg x \lor z) \land (z \lor w) \land x \land (y \lor v)$$

$\phi$ has 6 models.

Truth table method is $\theta(2^n)$.

DPLL method is $O(2^n)$, but is faster in practice.\(^1\)

A formula over the theory of strings can involve

- Word Equations: $X \circ U = Y \circ Z$
- Length Constraints: $4 < \text{Length}(X) < 10$
- Regular Language Membership: $X \in (a|b)^*$
- and more complex constraints: $(X = \text{substring}(Y, i, j), \ldots)$
Model Counting Strings

\[ X \in (0|(1(01^*0)*1))^* \]

Q: How many solutions for \( X \)? A: Infinitely many!

Q: How many solutions for \( X \) of length \( k \)?

A counting sequence for language \( L \) encodes

\[
 a_k = |\{ s : s \in L, \text{len}(s) = k \}| 
\]

\[
 a_0 = 1, a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 3, a_5 = 5, \ldots 
\]

<table>
<thead>
<tr>
<th>( k )</th>
<th>( X )</th>
<th>( a_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \varepsilon )</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1001, 1100, 1111</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10010, 10101, 11000, 11011, 11110</td>
<td>5</td>
</tr>
</tbody>
</table>

Outline

- Motivation and Background
- Model Counting Boolean Formulas
- String Model Counting
  - Automata-Based Methods
  - Non-Automata-Based Method
- String Model Counting Benchmarks
Deterministic Finite Automata

\[ X \in (0|1(01^*0)^*1)^* \]

\[ \{ s : s \in \mathcal{L}, \text{len}(s) = k \} \equiv \{ \pi : \pi \text{ is accepting path of length } k \} \]

String Counting \equiv Path Counting

How to count paths of length \( k \)?

**Dynamic Programming**

Initial Conditions

\[
\begin{align*}
a_0(0) &= 1, \quad a_0(1) = 0, \quad a_0(2) = 0
\end{align*}
\]

System of Recurrences

\[
\begin{align*}
a_0(k) &= a_0(k - 1) + a_1(k - 1) \\
a_1(k) &= a_0(k - 1) + a_2(k - 1) \\
a_2(k) &= a_1(k - 1) + a_2(k - 1)
\end{align*}
\]
Deterministic Finite Automata

How to count paths of length $k$?

**Matrix Exponentiation**

System of Recurrences

$$
\begin{align*}
a_0(k) &= a_0(k-1) + a_1(k-1) \\
a_1(k) &= a_0(k-1) + a_2(k-1) \\
a_2(k) &= a_1(k-1) + a_2(k-1)
\end{align*}
$$

$$
\begin{pmatrix}
a_0(k) \\
a_1(k) \\
a_2(k)
\end{pmatrix} =
\begin{pmatrix}
1 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
a_0(k-1) \\
a_1(k-1) \\
a_2(k-1)
\end{pmatrix}
$$

$$
a_k = (A^k)_{0,F}
$$

$$
a_4 = (A^4)_{0,0} = 3
$$

**Generating functions** are a way to compactly represent (possibly infinite) sequences.

$$
g(z) = \frac{1}{(1-z)^3} = \sum_{k=0}^{\infty} a_k z^k
$$

$$
g(z) = 1 z^0 + 3 z^1 + 6 z^2 + 10 z^3 + 15 z^4 + \ldots
$$

$$
g(z) = a_0 z^0 + a_1 z^1 + a_2 z^2 + a_3 z^3 + a_4 z^4 + \ldots
$$

Sequence element $a_k$ is the $k^{th}$ Taylor series coefficient of $g(z)$. 

$X \in (0|(1(01^*0)^*1))^*$

A generating function for language $L$ encodes

$$a_k = |\{s : s \in L, \text{len}(s) = k\}|$$

$$g(z) = 1z^0 + 1z^1 + 1z^2 + 1z^3 + 3z^4 + 5z^5 + \ldots$$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$X$</th>
<th>$a_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\varepsilon$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1001, 1100, 1111</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10010, 10101, 11000, 11011, 11110</td>
<td>5</td>
</tr>
</tbody>
</table>

Deterministic Finite Automata

How to count paths of length $k$?

Generating Functions

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$g(z) = \frac{\det(I - zA : i,j)}{\det(I - zA)}$$

$$g(z) = \frac{1 - z - z^2}{(z - 1)(2z^2 + z - 1)}$$

$$g(z) = 1z^0 + 1z^1 + 1z^2 + 1z^3 + 3z^4 + 5z^5 + \ldots$$
Automata-Based Model Counter (ABC)

CAV 2015: Automata-Based Model Counting for String Constraints. Abdulbaki Aydin, Lucas Bang, Tevfik Bultan:

Idea: Convert string constraints to DFA. Count paths in DFA.
Password Changing Policy

Constraint on NEW_P

(declare-fun NEW_P () String)
(not (contains (toLower NEW_P) "abc-16"))
(not (contains "abc-16" (toLower NEW_P)))
(not (contains (toLower NEW_P) "61-cba"))
(not (contains "61-cba" (toLower NEW_P)))

(check-sat)
(model-count)

Figure: Solution DFA for all possible values of NEWP.
Password Changing Policy

Figure: Transition matrix for DFA for all possible values of NEWP.
Password Changing Policy

Generating function which enumerates NEW_P:

\[
g(z) = \frac{8096z^{12} - 8128z^{11} + 32z^{10} + 16z^{7} - 16z^{6} - 256z^{2} + 257z - 1}{194304z^{17} + 225920z^{16} + 241984z^{15} + \ldots + z^{5} - 6114z^{4} - 2280z^{3} - 247z^{2}}
\]

\[
g(z) = 247z^{2} + 65759z^{3} + 16842945z^{4} + 4311810213z^{5} + 1103823437965z^{6} + \ldots
\]

To answer our quantitative question:

- Brute force searching for password length = 6: \(256^6 = 2^{48}\) passwords.
- If adversary knows old_p and the policy: \(1103823437965 \approx 2^{40.0056}\) passwords.
- Reduces search space by about factor of \(2^{7.9944}\)

Outline

- Motivation and Background
- Model Counting Boolean Formulas
- String Model Counting
  - Automata-Based Methods
  - Non-Automata-Based Method
- String Model Counting Benchmarks

31 / 45

32 / 45
SMC Model Counting


SMC Tool Online: https://github.com/loiluu/smc

Idea: go directly from constraints to $g(z)$ using transformations.

For a regular expression constraint, generating function can be derived recursively.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>$1z^0$</td>
</tr>
<tr>
<td>$c$</td>
<td>$1z^1$</td>
</tr>
<tr>
<td>$A</td>
<td>B$</td>
</tr>
<tr>
<td>$A \circ B$</td>
<td>$A(z) \times B(z)$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$1/(1 - A(z))$</td>
</tr>
</tbody>
</table>
Regular Expressions

\[ X \in (0|(1(01*0)*1))^* \]

Generating Function:

\[ g(z) = \frac{1}{1-z-z^2} \]

\[ = \frac{1-z-z^2}{(z-1)(2z^2+z-1)} \]

\[ g(z) = 1z^0 + 1z^1 + 1z^2 + 1z^3 + 3z^4 + 5z^5 + \ldots \]

Other operations in SMC

Specialized transformations for other operations

\[ \text{contains}(s_1, s_2) \mapsto \frac{z^n}{(1-Mz)(z^n+(1-Mz)c(z))} \]

\[ F_1 \lor F_2 \mapsto [\max(L_1(z), L_2(z)), \min(U_1(z) + U_2(z), G(z))] \]

Also handle substring, length, negation, conjunction, \ldots, with upper and lower bounds.
Experimental Comparison

Table: Log scaled comparison between SMC and ABC

<table>
<thead>
<tr>
<th></th>
<th>bound</th>
<th>SMC lower bound</th>
<th>SMC upper bound</th>
<th>ABC count</th>
</tr>
</thead>
<tbody>
<tr>
<td>nullhttpd</td>
<td>500</td>
<td>3752</td>
<td>3760</td>
<td>3760</td>
</tr>
<tr>
<td>ghttpd</td>
<td>620</td>
<td>4880</td>
<td>4896</td>
<td>4896</td>
</tr>
<tr>
<td>csplit</td>
<td>629</td>
<td>4852</td>
<td>4921</td>
<td>4921</td>
</tr>
<tr>
<td>grep</td>
<td>629</td>
<td>4676</td>
<td>4763</td>
<td>4763</td>
</tr>
<tr>
<td>wc</td>
<td>629</td>
<td>4281</td>
<td>4284</td>
<td>4281</td>
</tr>
<tr>
<td>obscure</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
### JavaScript Benchmarks

- Kaluza benchmarks, extracted from JavaScript code via DSE, [Saxena, SSP 2010]
  - Small Constraints (19,731):
    - ABC: 19,731 constraints, average 0.32 seconds per constraint
    - SMC: 17,559 constraints, average 0.26 seconds per constraint.
  - Big Constraints (1,587):
    - ABC: 1,587 constraints, average 0.34 seconds per constraint
    - SMC: 1,342 constraints, average 5.29 seconds per constraint

---

### ABC Bonus: Model Counting Linear Integer Arithmetic

What is this language?

\[ X \in (0|(1(01*0)*1))^* \]

\[ L(X) = \{s | s \text{ is a binary number divisible by 3}\} \]

![DFA Diagram]

**Idea:** DFA can represent (some) relations on sets of binary integers. We can use similar techniques that we used for \#String to solve \#LIA.
Quantifier-Free Linear Integer Arithmetic \((\mathbb{Z}, +, <)\).

Constraints of the form:

\[ Ax < B, x \in \mathbb{Z}^n \]

It is possible to represent the solutions to a set of LIA constraints as a binary multi-track DFA.

**Binary Multi-track DFA**

**Solution DFA for LIA constraints.**

- Read bits of \( x \) and \( y \) from most to least significant.
- Alphabet is a tuple of bits: \( (b_x, b_y) \)

**Solution DFA for the constraint \( x > y \).**

Solutions of length \( n \equiv \) solutions within bound \( 2^n \)
Model Counting Summary

Counting Techniques for Different Theories

- **Boolean**
  - Truth Table (Brute Force)
  - DPLL
- **Strings**
  - DFA with Dynamic Programming, Matrix Multiplication, GFs
  - Regular Expression with GFs
- **Linear Integer Arithmetic**
  - Binary Multi-track DFA

Related work on model counting

- Pugh. Counting Solutions to Presburger Formulas: How and Why. 1994
- Parker. An Automata-Theoretic Algorithm for Counting Solutions to Presburger Formulas. Compiler Construction 2004
- Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. Mathematics of Operations Research 1994
- De Loerab. Effective lattice point counting in rational convex polytopes. JSC 2004
- Verdoolaege. Counting integer points in parametric polytopes using Barvinok's Rational Functions. 2007
- Kopf Symbolic Polytopes for Quantitative Interpolation and Verification. CAV 2015
- Luu. A Model Counter For Constraints Over Unbounded Strings. PLDI 2014
- Ravikumara. Weak minimization of DFA - an algorithm and applications. Implementation and Application of Automata 2004
- Chomsky. The Algebraic Theory of Context-Free Languages. 1963
- Birnbaum. The good old Davis-Putnam procedure helps counting models. JAIR 1999
Thank you.