Automata-Based String Analysis
Model Counting

Tevfik Bultan, Abdulbaki Aydin, Lucas Bang

Verification Laboratory
http://vlab.cs.ucsb.edu
Department of Computer Science
Overview
Overview

String Constraints
Overview

String Constraints → Model Counter

Number of Solutions: 2 / 46
Overview

- **String Constraints** → **Model Counter** → **Number of Solutions**

- Number of Solutions: 2 / 46
Can you solve it, Will Hunting?
Can you solve it, Will Hunting?

Given the graph, find:
1) the adjacency matrix $A$
2) the matrix giving the number of 3-step walks
3) the generating function for walks from point $i \rightarrow j$
4) the generating function for walks from points $1 \rightarrow 3$
Motivation and Background
Model Counting Boolean Formulas
String Model Counting
  - Automata-Based Methods
  - Non-Automata-Based Method
String Model Counting Benchmarks
A Motivating Example

An adversary learns a password. User must select a new password.
A Motivating Example

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

Policy for selecting a new password.
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 $\text{old}_p = "abc-16"$
A Motivating Example

Suppose an adversary knows $old_p = "abc-16"$ and knows the policy.
A Motivating Example

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

Constraints on possible values of NEW\_P

\[
\begin{align*}
\text{(not (contains (toLowerCase NEW\_P) "abc-16"))} \\
\text{(not (contains (toLowerCase NEW\_P) "61-cba"))} \\
\text{(not (contains "abc-16" (toLowerCase NEW\_P)))} \\
\text{(not (contains "61-cba" (toLowerCase NEW\_P)))}
\end{align*}
\]
A Motivating Example

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

### Constraints on possible values of NEW_P

- \( \text{not (contains (toLower NEW_P) "abc-16")} \)
- \( \text{not (contains (toLower NEW_P) "61-cba")} \)
- \( \text{not (contains "abc-16" (toLower NEW_P))} \)
- \( \text{not (contains "61-cba" (toLower NEW_P))} \)

If password length = \( n \), then there are \( |\Sigma|^n \) possible passwords.
A Motivating Example

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

Constraints on possible values of NEW_P

- $(\neg \text{contains (toLower NEW_P)} \text{"abc-16"}))$
- $(\neg \text{contains (toLower NEW_P)} \text{"61-cba"}))$
- $(\neg \text{contains "abc-16" (toLower NEW_P)})$
- $(\neg \text{contains "61-cba" (toLower NEW_P)})$

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

If adversary knows $old_p$ and the policy . . .

- how much is the reduction in search space?
- what is the probability of guessing the new password?
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...
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
Motivation

Techniques for model counting for other theories

<table>
<thead>
<tr>
<th>Boolean Logic Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ DPLL</td>
</tr>
<tr>
<td>▶ Random sampling based</td>
</tr>
<tr>
<td>▶ Approximations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linear Integer Arithmetic:</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ LattE</td>
</tr>
<tr>
<td>▶ Barvinok</td>
</tr>
</tbody>
</table>
Motivation

String manipulating programs are pervasive

- security critical functions,
- server side sanitization functions,
- databases,
- dynamic code generation.
## 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.
**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 ]
- S3# [Trinh, et. al. CAV 2017]
Outline

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

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

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

The **model counting problem**

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

how many models are there for $\phi$?
The **model counting problem**

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

how many models are there for $\phi$?

**Difficulty of Model Counting**

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

$|\phi| > 0 \iff \phi$ is satisfiable
### Model Counting Boolean SAT

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>v</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

φ = (x ∨ y) ∧ (¬x ∨ z) ∧ (z ∨ w) ∧ x ∧ (y ∨ v)

φ has 6 models.

Truth table method is $θ(2^n)$.

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

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

\(\phi\) has 6 models.
Model Counting Boolean SAT

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>v</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

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

\( \phi \) has 6 models.

Truth table method is \( \theta(2^n) \).
Model Counting Boolean SAT

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>v</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

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

$\phi$ has 6 models.

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

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

\(^1\) Birnbaum, et. al. The good old Davis-Putnam procedure helps counting models. JAIR 1999.
Outline

- Motivation and Background
- Model Counting Boolean Formulas
- String Model Counting
  - Automata-Based Methods
  - Non-Automata-Based Method
- String Model Counting Benchmarks
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: \( (X = \text{substring}(Y, i, j), \ldots) \)
A formula over the theory of strings can involve

- Word Equations: $X \circ U = Y \circ Z$
A formula over the theory of strings can involve

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

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

Q: How many solutions for \( X \)?

\[ a_0 = 1, \quad a_1 = 1, \quad a_2 = 1, \quad a_3 = 1, \quad a_4 = 3, \quad a_5 = 5, \ldots \]
Model Counting Strings

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

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

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

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

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

\[ a_0 = 1 \]

<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>
</tbody>
</table>
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 \( \mathcal{L} \) encodes

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

\[ a_0 = 1, \ a_1 = 1 \]

\[
\begin{array}{ccc}
  k & X & a_k \\
  0 & \varepsilon & 1 \\
  1 & 0 & 1 \\
\end{array}
\]
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 \( \mathcal{L} \) encodes

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

\[ a_0 = 1, \quad a_1 = 1, \quad a_2 = 1 \]

<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>
</tbody>
</table>
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 \( \mathcal{L} \) encodes

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

\( a_0 = 1, a_1 = 1, a_2 = 1, a_3 = 1 \)

<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>
</tbody>
</table>
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 \( \mathcal{L} \) encodes

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

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

<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>
</tbody>
</table>
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 \( \mathcal{L} \) encodes

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

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

<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
Deterministic Finite Automata

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

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

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

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

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

\[
\begin{align*}
|\{ s : s \in \mathcal{L}, \text{len}(s) = k \}| & \equiv |\{ \pi : \pi \text{ is accepting path of length } k \}| \\
\text{String Counting} & \equiv \text{Path Counting}
\end{align*}
\]
How to count paths of length $k$?
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**
Deterministic Finite Automata

How to count paths of length $k$?

Dynamic Programming

$a_k$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

$a_k(s) =$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

$$a_k(s) = a_{k-1}(s')$$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

$$a_k(s) = a_{k-1}(s')$$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

$$a_k(s) = a_{k-1}(s')$$
Deterministic Finite Automata

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

**Dynamic Programming**

\[
a_k(s) = \sum_{s' \to s} a_{k-1}(s')
\]
Deterministic Finite Automata

How to count paths of length $k$?

Dynamic Programming

Initial Conditions

$$a_k(s) = \sum_{s' \rightarrow s} a_{k-1}(s')$$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

Initial Conditions

$a_0(0) = 1$

$$a_k(s) = \sum_{s' \rightarrow s} a_{k-1}(s')$$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

Initial Conditions

$$a_0(0) = 1, a_0(1) = 0, a_0(2) = 0$$

$$a_k(s) = \sum_{s' \rightarrow s} a_{k-1}(s')$$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

Initial Conditions

$$a_0(0) = 1, \ a_0(1) = 0, \ a_0(2) = 0$$

System of Recurrences

$$a_k(0) = a_{k-1}(0) + a_{k-1}(1)$$
Deterministic Finite Automata

How to count paths of length $k$?

**Dynamic Programming**

Initial Conditions

$$a_0(0) = 1, \ a_0(1) = 0, \ a_0(2) = 0$$

System of Recurrences

$$a_k(0) = a_{k-1}(0) + a_{k-1}(1)$$
$$a_k(1) = a_{k-1}(0) + a_{k-1}(2)$$
$$a_k(2) = a_{k-1}(1) + a_{k-1}(2)$$
Deterministic Finite Automata

How to count paths of length $k$?

Matrix Exponentiation

System of Recurrences

$$a_k(0) = a_{k-1}(0) + a_{k-1}(1)$$

$$a_k(1) = a_{k-1}(0) + a_{k-1}(2)$$

$$a_k(2) = a_{k-1}(1) + a_{k-1}(2)$$

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

$$a_k = (A^k)_{0,0}$$
Deterministic Finite Automata

How to count paths of length $k$?

System of Recurrences

\[
\begin{align*}
a_k(0) &= a_{k-1}(0) + a_{k-1}(1) \\
a_k(1) &= a_{k-1}(0) + a_{k-1}(2) \\
a_k(2) &= a_{k-1}(1) + a_{k-1}(2)
\end{align*}
\]
Deterministic Finite Automata

How to count paths of length $k$?

System of Recurrences

\[
\begin{align*}
a_k(0) &= a_{k-1}(0) + a_{k-1}(1) \\
a_k(1) &= a_{k-1}(0) + a_{k-1}(2) \\
a_k(2) &= a_{k-1}(1) + a_{k-1}(2)
\end{align*}
\]

\[
\begin{pmatrix}
a_0(k) \\
a_1(k) \\
a_2(k)
\end{pmatrix} =
\begin{pmatrix}
1 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
a_0(k-1) \\
a_1(k-1) \\
a_2(k-1)
\end{pmatrix}
\]
Deterministic Finite Automata

How to count paths of length $k$?

**Matrix Exponentiation**

System of Recurrences

\[
\begin{align*}
a_k(0) &= a_{k-1}(0) + a_{k-1}(1) \\
a_k(1) &= a_{k-1}(0) + a_{k-1}(2) \\
a_k(2) &= a_{k-1}(1) + a_{k-1}(2)
\end{align*}
\]

\[
\begin{pmatrix}
a_0(k) \\
a_1(k) \\
a_2(k)
\end{pmatrix}
= 
\begin{pmatrix}
1 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 1
\end{pmatrix}
^k
\begin{pmatrix}
a_0(k-1) \\
a_1(k-1) \\
a_2(k-1)
\end{pmatrix}
\]
How to count paths of length $k$?

**Matrix Exponentiation**

System of Recurrences

$$a_k(0) = a_{k-1}(0) + a_{k-1}(1)$$
$$a_k(1) = a_{k-1}(0) + a_{k-1}(2)$$
$$a_k(2) = a_{k-1}(1) + a_{k-1}(2)$$

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

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

How to count paths of length $k$?

**Matrix Exponentiation**

System of Recurrences

$$a_k(0) = a_{k-1}(0) + a_{k-1}(1)$$
$$a_k(1) = a_{k-1}(0) + a_{k-1}(2)$$
$$a_k(2) = a_{k-1}(1) + a_{k-1}(2)$$

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

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

$$a_4 = (A^4)_{0,0} = 3$$
Generating functions are a way to compactly represent (possibly infinite) sequences. 

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

Sequence element \( a_k \) is the \( k \)th Taylor series coefficient of \( g(z) \).
Generating functions are a way to compactly represent (possibly infinite) sequences.
Generating functions are a way to compactly represent (possibly infinite) sequences.

\[ g(z) = \frac{1}{(1 - z)^3} \]
Generating functions are a way to compactly represent (possibly infinite) sequences.

\[
g(z) = \frac{1}{(1 - z)^3} = \sum_{k=0}^{\infty} a_k z^k
\]
Generating functions are a way to compactly represent (possibly infinite) sequences.

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

\[ g(z) = 1z^0 + 3z^1 + 6z^2 + 10z^3 + 15z^4 + \ldots \]
Generating functions are a way to compactly represent (possibly infinite) sequences.

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

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

\[ g(z) = a_0 z^0 + a_1 z^1 + a_2 z^2 + a_3 z^3 + a_4 z^4 + \ldots \]
**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 \( g(z) \).
The Taylor series of a function $g(z)$ that is differentiable at 0 is the power series

$$g(0) + \frac{g'(0)}{1!} x + \frac{g''(0)}{2!} x^2 + \frac{g'''(0)}{3!} x^3 + \cdots.$$
The Taylor series of a function $g(z)$ that is differentiable at 0 is the power series

$$g(0) + \frac{g'(0)}{1!} x + \frac{g''(0)}{2!} x^2 + \frac{g'''(0)}{3!} x^3 + \cdots.$$ 

which can be written in the more compact sigma notation as

$$\sum_{n=0}^{\infty} \frac{g^{(n)}(a)}{n!} (x - a)^n$$

where $n!$ denotes the factorial of $n$ and $g^{(n)}(a)$ denotes the $n$-th derivative of $f$ evaluated at the point $a$. 


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

A generating function for language \( \mathcal{L} \) encodes

\[ a_k = |\{s : s \in \mathcal{L}, \text{len}(s) = k\}| \]
\[ X \in (0|(1(01^*0)^*1))^* \]

A generating function for language \( \mathcal{L} \) encodes

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

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

A generating function for language \( \mathcal{L} \) encodes

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

\[ g(z) = z^0 \]

<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>
</tbody>
</table>
\[ X \in (0|(1(01^*0)^*1))^* \]

A generating function for language \( \mathcal{L} \) encodes

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

\[ g(z) = 1z^0 + 1z^1 \]

<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>
</tbody>
</table>
\[ X \in (0|(1(01*0)*1))^{*} \]

A generating function for language \( \mathcal{L} \) encodes

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

\[ g(z) = 1z^0 + 1z^1 + 1z^2 \]

<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>
</tbody>
</table>
\[ X \in (0|(1(01\ast0)^*1))^* \]

A generating function for language \( \mathcal{L} \) encodes

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

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

<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>
</tbody>
</table>
\[ X \in (0|1(01^*0)^*1))^* \]

A generating function for language \( \mathcal{L} \) encodes

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

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

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \mathcal{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>
</tbody>
</table>
$X \in (0\{(101*0)*1\})^*$

A generating function for language $\mathcal{L}$ encodes

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

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 $k$?

\[
A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}
\]

\[
g(z) = \det(I - zA)_{i,j} (-1)^{n-1} \det(I - zA) = 1 - z - z^2 (z - 1) (2z^2 + z - 1)
\]

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

How to count paths of length $k$?

**Generating Functions**

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

$$g(z) = \frac{\det(I - zA : i, j)}{(-1)^n \det(I - zA)}$$
Deterministic Finite Automata

How to count paths of length $k$?

**Generating Functions**

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

$$g(z) = \frac{\det(I - zA : i, j)}{(-1)^n \det(I - zA)}$$

$$g(z) = \frac{1 - z - z^2}{(z - 1)(2z^2 + z - 1)}$$
How to count paths of length $k$?

**Generating Functions**

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

$g(z) = \frac{\det(I - zA : i, j)}{(-1)^n \det(I - zA)}$

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

$g(z) = 1z^0 + 1z^1 + 1z^2 + 1z^3 + 3z^4 + 5z^5 + \ldots$
Good job, Will Hunting!!!

This is correct. Who did this?
Automata-Based Model Counter (ABC)

Automata-Based Model Counter (ABC)


String Constraints $\rightarrow$ ABC $\rightarrow$ Counting Function $f(k)$
Automata-Based Model Counter (ABC)


String Constraints $\rightarrow$ ABC $\rightarrow$ Counting Function $f(k)$

String Length, $k$
Automata-Based Model Counter (ABC)


String Constraints $\rightarrow$ ABC $\rightarrow$ Counting Function $f(k)$

String Length, $k$ $\downarrow$

Number of solutions of length $k$
Automata-Based Model Counter (ABC)


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)
Password Changing Policy

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

Figure: Transition matrix for DFA for all possible values of NEWP.
Generating function which enumerates NEW_P:

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

Generating function which enumerates NEW_P:

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

\[ g(z) = 247z^2 + 65759z^3 + 16842945z^4 + 4311810213z^5 + 1103823437965z^6 + \ldots \]
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.
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 \(\text{old}_p\) and the policy: \(1103823437965 \approx 2^{40.0056}\) passwords.
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 \(\text{old}_p\) 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
SMC Model Counting


SMC Tool Online: https://github.com/loiluu/smc
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.
For a regular expression constraint, generating function can be derived recursively.

\[ \varepsilon \mapsto 1z^0 \]
For a regular expression constraint, generating function can be derived recursively.

\[
\varepsilon \mapsto 1z^0 \\
C \mapsto 1z^1
\]
SMC Model Counting

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$1z^0$</td>
</tr>
<tr>
<td>$c$</td>
<td>$1z^1$</td>
</tr>
<tr>
<td>$A</td>
<td>B$</td>
</tr>
</tbody>
</table>
SMC Model Counting

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

\[ \varepsilon \mapsto 1z^0 \]
\[ c \mapsto 1z^1 \]
\[ A|B \mapsto A(z) + B(z) \]
\[ A \circ B \mapsto A(z) \times B(z) \]
For a regular expression constraint, generating function can be derived recursively.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Corresponding Generating Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</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

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

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

Diagram:

```
*   
|----|  
V   
|----|  
  0  
|----|  
   1  
|----|  
      *  
|----|   
0   
|----|  
   0  
|----|  
      *  
|----|   
      1  
|----|  
   1  
```

Generating Function:

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

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

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

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

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

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

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

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

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

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

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

\[ X \in (0|(101*0)*1))* \]

Generating Function:

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

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

Generating Function:

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

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

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

Generating Function:

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

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

Specialized transformations for other operations

\[ (s_1, s_2) \mapsto z_n (1 - Mz) (z + (1 - Mz)c(z)) \]

\[ F_1 \lor F_2 \mapsto \begin{cases} \max (L_1(z), L_2(z)), \\ \min (U_1(z) + U_2(z), G(z)) \end{cases} \]

Also handle substring, length, negation, conjunction, …, with upper and lower bounds.
Other operations in SMC

<table>
<thead>
<tr>
<th>Specialized transformations for other operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{contains}(s_1, s_2) \mapsto \frac{z^n}{(1-Mz)(z^n + (1-Mz)c(z))} )</td>
</tr>
</tbody>
</table>
Other operations in SMC

<table>
<thead>
<tr>
<th>Operation</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contains(s_1, s_2)</code></td>
<td>$\mapsto (1-Mz)(z^n + (1-Mz)c(z))$</td>
</tr>
<tr>
<td>$F_1 \lor F_2$</td>
<td>$\mapsto [\max(L_1(z), L_2(z)), \min(U_1(z) + U_2(z), G(z))]$</td>
</tr>
</tbody>
</table>
Other operations in SMC

Specialized transformations for other operations

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

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

Also handle substring, length, negation, conjunction, ... , with upper and lower bounds.
Outline

- Motivation and Background
- Model Counting Boolean Formulas
- String Model Counting
  - Automata-Based Methods
  - Non-Automata-Based Method
- String Model Counting Benchmarks
### 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>
### Experimental Comparison

#### JavaScript Benchmarks

- Kaluza benchmarks, extracted from JavaScript code via DSE, [Saxena, SSP 2010]

<table>
<thead>
<tr>
<th>Constraints</th>
<th>ABC</th>
<th>SMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>19,731</td>
<td>17,559</td>
</tr>
<tr>
<td>Average per constraint</td>
<td>0.32 seconds</td>
<td>0.26 seconds</td>
</tr>
<tr>
<td>Big</td>
<td>1,587</td>
<td>1,342</td>
</tr>
<tr>
<td>Average per constraint</td>
<td>0.34 seconds</td>
<td>5.29 seconds</td>
</tr>
</tbody>
</table>
## Experimental Comparison

### 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.
## 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
What is this language?

\[ X \in (0 | (10) \cdot 0) \cdot 1 \cdot L(X) = \{ s | s \text{ is a binary number divisible by } 3 \} \]

Idea: DFA can represent (some) relations on sets of binary integers. We can use similar techniques that we used for \#String to solve \#LIA.
What is this language?

\[ X \in (0|(1(01*0)*1))\]
What is this language?

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

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

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

\( L(X) = \{s | s \text{ is a binary number divisible by } 3\} \)
What is this language?

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

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

**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 (\(\mathbb{Z}, +, <\)).
Quantifier-Free Linear Integer Arithmetic ($\mathbb{Z}, +, <$).

Constraints of the form:

$$Ax < B, x \in \mathbb{Z}^n$$
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: \( \begin{pmatrix} b_x \\ b_y \end{pmatrix} \)

Solution DFA for the constraint $x > y$.

\[
\begin{align*}
(0, 0), (1, 0) & \quad \rightarrow \quad (0, 0), (0, 1), (1, 1) \\
1 & \quad \rightarrow \quad (0, 0), (0, 1), (1, 1), (1, 1) \\
0 & \quad \rightarrow \quad (0, 0), (0, 1), (1, 0), (1, 1) \\
0 & \quad \rightarrow \quad (0, 0), (0, 1), (1, 0), (1, 1)
\end{align*}
\]
Solution DFA for LIA constraints.

- Read bits of $x$ and $y$ from most to least significant.
- Alphabet is a tuple of bits: $\begin{pmatrix} b_x \\ b_y \end{pmatrix}$

Solution DFA for the constraint $x > y$.

Solutions of length $n \equiv$ solutions within bound $2^n$
## Counting Techniques for Different Theories

- **Boolean**

---

/model COUNTING SUMMARY

Counting Techniques for Different Theories

- Boolean
## Model Counting Summary

### Counting Techniques for Different Theories

- **Boolean**
  - Truth Table (Brute Force)
  - DPLL
<table>
<thead>
<tr>
<th>Counting Techniques for Different Theories</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Boolean</td>
</tr>
<tr>
<td>▪ Truth Table (Brute Force)</td>
</tr>
<tr>
<td>▪ DPLL</td>
</tr>
<tr>
<td>• Strings</td>
</tr>
<tr>
<td>▪ DFA with Dynamic Programming, Matrix Multiplication, GFs</td>
</tr>
<tr>
<td>▪ Regular Expression with GFs</td>
</tr>
</tbody>
</table>
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

- Sedgewick. Analytic Combinatorics Chapter 5: Generating Functions. 2009
- 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 Loera. 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.