PLDI 2016 Tutorial
Automata-Based String Analysis
Model Counting

Tevfik Bultan, Abdulbaki Aydin, Lucas Bang

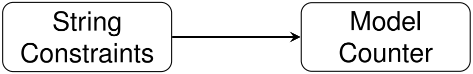
Verification Laboratory
http://vlab.cs.ucsb.edu
Department of Computer Science

Overview

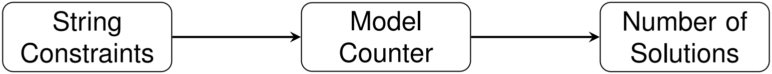
Overview

String
Constraints

Overview

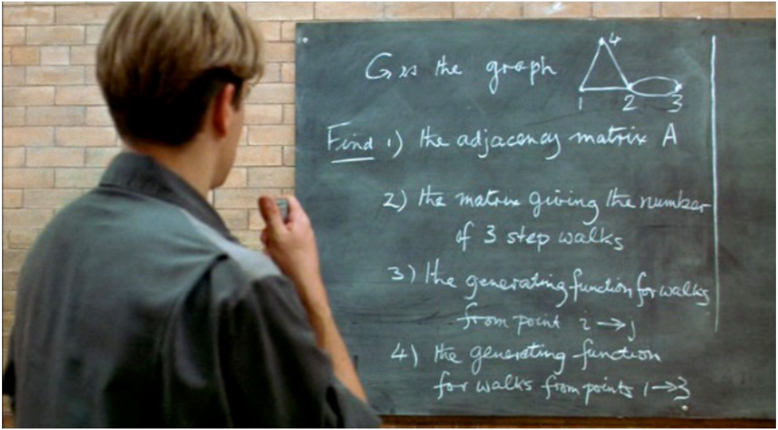


Overview



Can you solve it, Will Hunting?

Can you solve it, Will Hunting?



Outline

- 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.

```
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 `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 `old_p = "abc-16"` *and* knows the policy.

Constraints on possible values of NEW_P

```
(not (contains (toLower NEW_P) "abc-16"))
(not (contains (toLower NEW_P) "61-cba"))
(not (contains "abc-16" (toLower NEW_P)))
(not (contains "61-cba" (toLower NEW_P)))
```

A Motivating Example

Suppose an adversary knows `old_p = "abc-16"`
and knows the policy.

Constraints on possible values of NEW_P

```
(not (contains (toLower NEW_P) "abc-16"))
(not (contains (toLower NEW_P) "61-cba"))
(not (contains "abc-16" (toLower NEW_P)))
(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 = "abc-16"` *and* knows the policy.

Constraints on possible values of NEW_P

```
(not (contains (toLower NEW_P) "abc-16"))
(not (contains (toLower NEW_P) "61-cba"))
(not (contains "abc-16" (toLower NEW_P)))
(not (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?

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...

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

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.

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]

Outline

- 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 ϕ 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 ϕ from propositional logic, is it possible to assign all variables the values T (true) or F (false) so that the formula is true?

Example:

$$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$$

Model Counting

Recall the classic (boolean) SAT problem

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

Example:

$$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$$

ϕ is satisfiable by setting

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

Model Counting

Recall the classic (boolean) SAT problem

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

Example:

$$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$$

ϕ is satisfiable by setting

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

A satisfying assignment is called a **model** for ϕ.

Model Counting

The **model counting problem**

Given a formula ϕ over some theory (Boolean, LIA, Strings, ...)

how many models are there for ϕ?

Model Counting

The **model counting problem**

Given a formula ϕ over some theory (Boolean, LIA, Strings, ...)

how many models are there for ϕ?

Difficulty of Model Counting

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

Model Counting

The **model counting problem**

Given a formula ϕ over some theory (Boolean, LIA, Strings, . . .)

how many models are there for ϕ?

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

x	y	z	w	v	F
F	F	F	F	F	F
⋮	⋮	⋮	⋮	⋮	⋮
T	F	F	T	T	F
T	F	T	F	F	F
T	F	T	F	T	T
T	F	T	T	F	F
T	F	T	T	T	T
T	T	F	F	F	F
T	T	F	F	T	F
T	T	F	T	F	F
T	T	F	T	T	F
T	T	T	F	F	T
T	T	T	F	T	T
T	T	T	T	F	T
T	T	T	T	T	T

$$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$$

Model Counting Boolean SAT

x	y	z	w	v	F
F	F	F	F	F	F
⋮	⋮	⋮	⋮	⋮	⋮
T	F	F	T	T	F
T	F	T	F	F	F
T	**F**	**T**	**F**	**T**	**T**
T	F	T	T	F	F
T	**F**	**T**	**T**	**T**	**T**
T	T	F	F	F	F
T	T	F	F	T	F
T	T	F	T	F	F
T	T	F	T	T	F
T	**T**	**T**	**F**	**F**	**T**
T	**T**	**T**	**F**	**T**	**T**
T	**T**	**T**	**T**	**F**	**T**
T	**T**	**T**	**T**	**T**	**T**

$$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$$

ϕ has 6 models.

Model Counting Boolean SAT

x	y	z	w	v	F
F	F	F	F	F	F
⋮	⋮	⋮	⋮	⋮	⋮
T	F	F	T	T	F
T	F	T	F	F	F
T	**F**	**T**	**F**	**T**	**T**
T	F	T	T	F	F
T	**F**	**T**	**T**	**T**	**T**
T	T	F	F	F	F
T	T	F	F	T	F
T	T	F	T	F	F
T	T	F	T	T	F
T	**T**	**T**	**F**	**F**	**T**
T	**T**	**T**	**F**	**T**	**T**
T	**T**	**T**	**T**	**F**	**T**
T	**T**	**T**	**T**	**T**	**T**

$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$

ϕ has 6 models.

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

Model Counting Boolean SAT

x	y	z	w	v	F
F	F	F	F	F	F
⋮	⋮	⋮	⋮	⋮	⋮
T	F	F	T	T	F
T	F	T	F	F	F
T	**F**	**T**	**F**	**T**	**T**
T	F	T	T	F	F
T	**F**	**T**	**T**	**T**	**T**
T	T	F	F	F	F
T	T	F	F	T	F
T	T	F	T	F	F
T	T	F	T	T	F
T	**T**	**T**	**F**	**F**	**T**
T	**T**	**T**	**F**	**T**	**T**
T	**T**	**T**	**T**	**F**	**T**
T	**T**	**T**	**T**	**T**	**T**

$\phi = (x \vee y) \wedge (\neg x \vee z) \wedge (z \vee w) \wedge x \wedge (y \vee v)$

ϕ 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

Model Counting Strings

A formula over the theory of strings can involve

Model Counting Strings

A formula over the theory of strings can involve

- Word Equations: $X \circ U = Y \circ Z$

Model Counting Strings

A formula over the theory of strings can involve

- Word Equations: $X \circ U = Y \circ Z$
- Length Constraints: $4 < Length(X) < 10$

Model Counting Strings

A formula over the theory of strings can involve

- Word Equations: $X \circ U = Y \circ Z$
- Length Constraints: $4 < \mathit{Length}(X) < 10$
- Regular Language Membership: $X \in (a|b)^*$

Model Counting Strings

A formula over the theory of strings can involve

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

Model Counting Strings

A formula over the theory of strings can involve

- Word Equations: $X \circ U = Y \circ Z$
- Length Constraints: $4 < \mathit{Length}(X) < 10$
- 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?

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 $\mathcal{L}$ 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

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

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

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

$a_0 = 1$

k	X	a_k
0	ε	1

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

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

$a_0 = 1, a_1 = 1$

k	X	a_k
0	ε	1
1	0	1

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

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

$a_0 = 1, a_1 = 1, a_2 = 1$

k	X	a_k
0	ε	1
1	0	1
2	11	1

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

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

$$a_0 = 1, a_1 = 1, a_2 = 1, a_3 = 1$$

k	X	a_k
0	ε	1
1	0	1
2	11	1
3	110	1

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

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

$$a_0 = 1, a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 3$$

k	X	a_k
0	ε	1
1	0	1
2	11	1
3	110	1
4	1001, 1100, 1111	3

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

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

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

k	X	a_k
0	ε	1
1	0	1
2	11	1
3	110	1
4	1001, 1100, 1111	3
5	10010, 10101, 11000, 11011, 11110	5

Outline

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

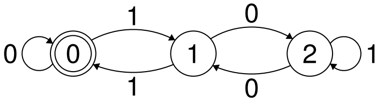
Deterministic Finite Automata

Deterministic Finite Automata

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

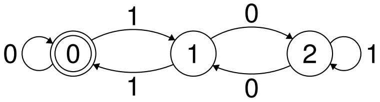
Deterministic Finite Automata

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



Deterministic Finite Automata

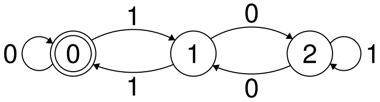
$$X \in (0|(1(01^*0)^*1))^*$$



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

Deterministic Finite Automata

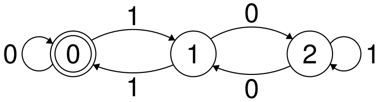
$$X \in (0|(1(01^*0)^*1))^*$$



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

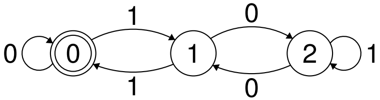
String Counting $\equiv$ Path Counting

Deterministic Finite Automata



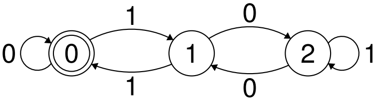
How to count paths of length k?

Deterministic Finite Automata



How to count paths of length k?
Dynamic Programming

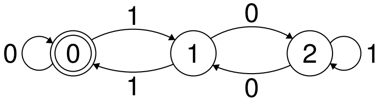
Deterministic Finite Automata



How to count paths of length k?
Dynamic Programming

a_k

Deterministic Finite Automata

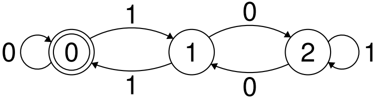


How to count paths of length k?

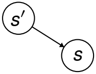
Dynamic Programming

$a_k(s) =$

Deterministic Finite Automata

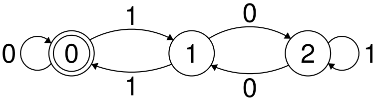


How to count paths of length k?
Dynamic Programming

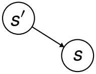


$$a_k(s) = \qquad a_{k-1}(s')$$

Deterministic Finite Automata

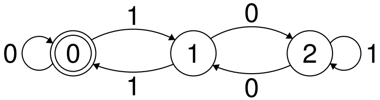


How to count paths of length k?
Dynamic Programming

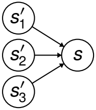


$$a_k(s) = \qquad a_{k-1}(s')$$

Deterministic Finite Automata

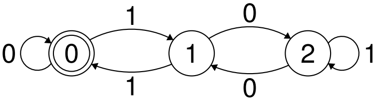


How to count paths of length k?
Dynamic Programming

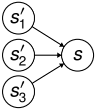


$$a_k(s) = \qquad a_{k-1}(s')$$

Deterministic Finite Automata

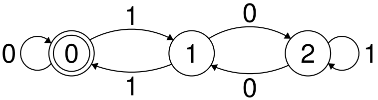


How to count paths of length k?
Dynamic Programming

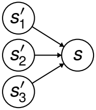


$$a_k(s) = \sum_{s' \to s} a_{k-1}(s')$$

Deterministic Finite Automata



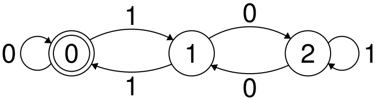
How to count paths of length k?
Dynamic Programming



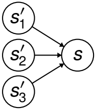
Initial Conditions

$$a_k(s) = \sum_{s' \to s} a_{k-1}(s')$$

Deterministic Finite Automata



How to count paths of length k?
Dynamic Programming

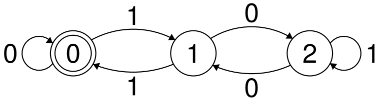


Initial Conditions

$$a_0(0) = 1$$

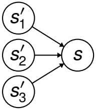
$$a_k(s) = \sum_{s' \to s} a_{k-1}(s')$$

Deterministic Finite Automata



How to count paths of length k?

Dynamic Programming

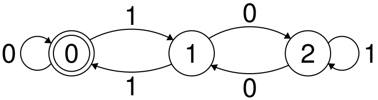


Initial Conditions

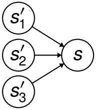
$$a_0(0) = 1, a_0(1) = 0, a_0(2) = 0$$

$$a_k(s) = \sum_{s' \to s} a_{k-1}(s')$$

Deterministic Finite Automata



How to count paths of length k?
Dynamic Programming



$$a_k(s) = \sum_{s' \to s} a_{k-1}(s')$$

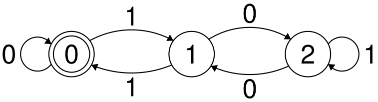
Initial Conditions

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

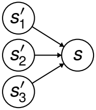
System of Recurrences

$$a_0(k) = a_0(k-1) + a_1(k-1)$$

Deterministic Finite Automata



How to count paths of length k?
Dynamic Programming



$$a_k(s) = \sum_{s' \to s} a_{k-1}(s')$$

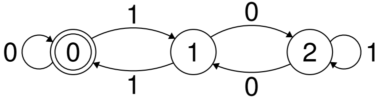
Initial Conditions

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

System of Recurrences

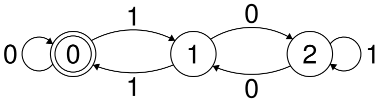
$$\begin{aligned} a_0(k) &= a_0(k-1) + a_1(k-1) \\ a_1(k) &= a_0(k-1) + a_2(k-1) \\ a_2(k) &= a_1(k-1) + a_2(k-1) \end{aligned}$$

Deterministic Finite Automata



How to count paths of length k?

Deterministic Finite Automata

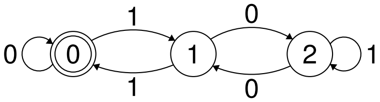


How to count paths of length k?

System of Recurrences

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

Deterministic Finite Automata



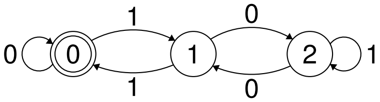
How to count paths of length k?

System of Recurrences

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

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

Deterministic Finite Automata



How to count paths of length k?
Matrix Exponentiation

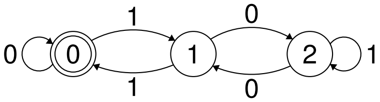
System of Recurrences

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

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

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

Deterministic Finite Automata



How to count paths of length k?
Matrix Exponentiation

System of Recurrences

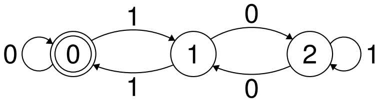
$$\begin{aligned}
a_0(k) &= a_0(k-1) + a_1(k-1) \\
a_1(k) &= a_0(k-1) + a_2(k-1) \\
a_2(k) &= a_1(k-1) + a_2(k-1)
\end{aligned}$$

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

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

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

Deterministic Finite Automata



How to count paths of length k?
Matrix Exponentiation

System of Recurrences

$$a_0(k) = a_0(k-1) + a_1(k-1)$$
$$a_1(k) = a_0(k-1) + a_2(k-1)$$
$$a_2(k) = a_1(k-1) + a_2(k-1)$$

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

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

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

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$g(z) = 1z^0$$

k	X	a_k
0	ε	1

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

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

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

$$g(z) = 1z^0 + 1z^1$$

k	X	a_k
0	ε	1
1	0	1

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

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

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

$$g(z) = 1z^0 + 1z^1 + 1z^2$$

k	X	a_k
0	ε	1
1	0	1
2	11	1

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

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

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

$$g(z) = 1z^0 + 1z^1 + 1z^2 + 1z^3$$

k	X	a_k
0	ε	1
1	0	1
2	11	1
3	110	1

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

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

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

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

k	X	a_k
0	ε	1
1	0	1
2	11	1
3	110	1
4	1001, 1100, 1111	3

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

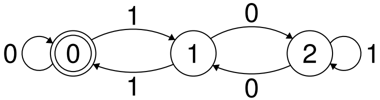
A generating function for language $\mathcal{L}$ encodes

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

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

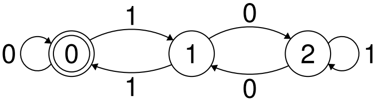
k	X	a_k
0	ε	1
1	0	1
2	11	1
3	110	1
4	1001, 1100, 1111	3
5	10010, 10101, 11000, 11011, 11110	5

Deterministic Finite Automata



How to count paths of length k?

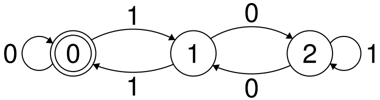
Deterministic Finite Automata



How to count paths of length k?
Generating Functions

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad g(z) = \frac{\det(I - zA : i, j)}{(-1)^n \det(I - zA)}$$

Deterministic Finite Automata



How to count paths of length k?

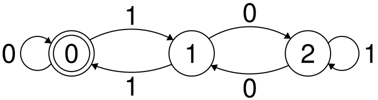
Generating Functions

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

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

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

Deterministic Finite Automata



How to count paths of length k?
Generating Functions

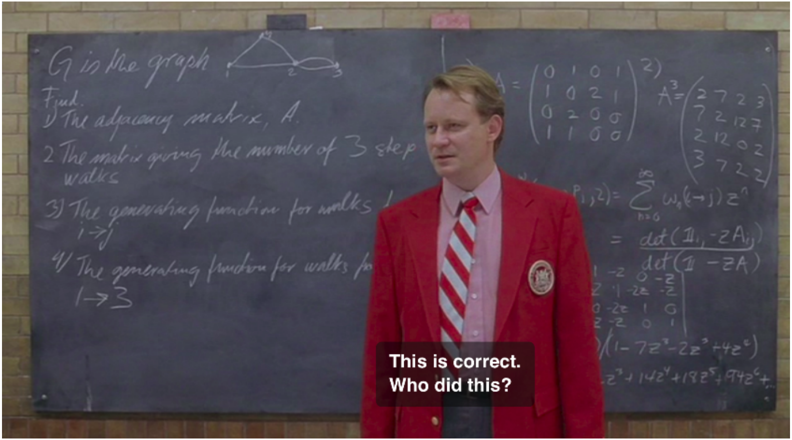
$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

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

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

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

Good job, Will Hunting!!!

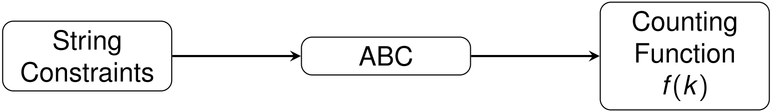


Automata-Based Model Counter (ABC)

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

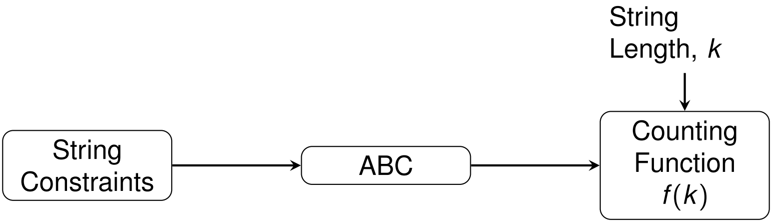
Automata-Based Model Counter (ABC)

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



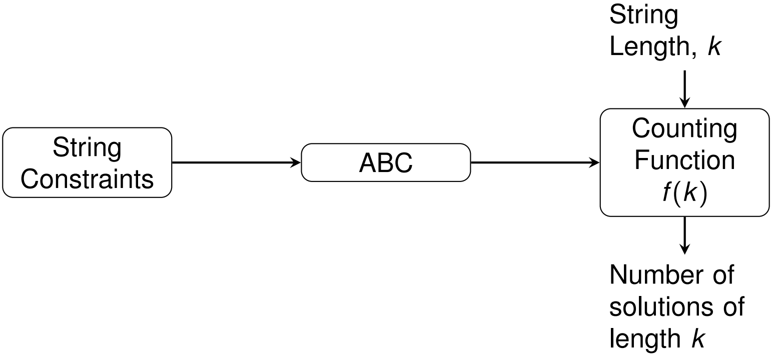
Automata-Based Model Counter (ABC)

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



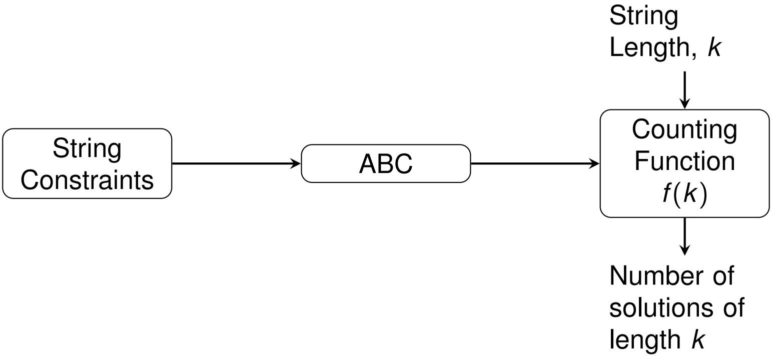
Automata-Based Model Counter (ABC)

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



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)
```

Password Changing Policy

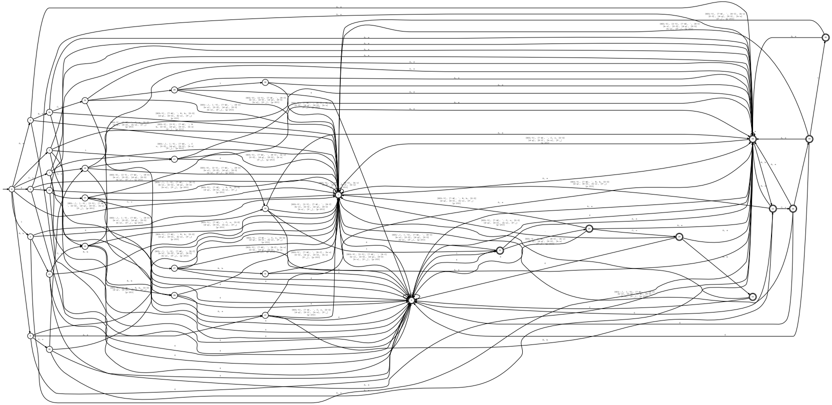


Figure : Solution DFA for all possible values of NEWP.

Password Changing Policy

$$
\begin{pmatrix}
0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

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

Password Changing Policy

$$
\begin{pmatrix}
0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

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

Password Changing Policy

Generating function which enumerates NEW_P:

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

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 `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 `old_p` and the policy: $1103823437965 \approx 2^{40.0056}$ passwords.
- Reduces search space by about factor of $2^{7.9944}$

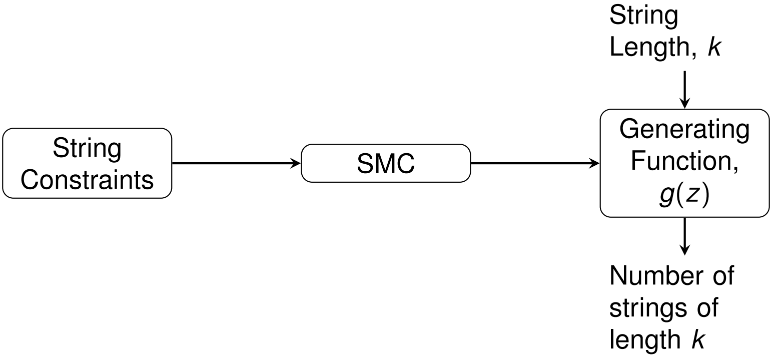
Outline

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

SMC Model Counting

PLDI 2014: A Model Counter For Constraints Over Unbounded Strings. Luu, Shinde, Saxena, Demsky.

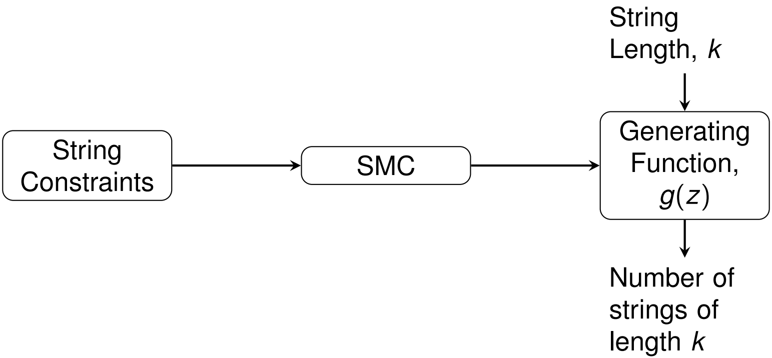
SMC Tool Online: https://github.com/loiluu/smc



SMC Model Counting

PLDI 2014: A Model Counter For Constraints Over Unbounded Strings. Luu, Shinde, Saxena, Demsky.

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



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

SMC Model Counting

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

SMC Model Counting

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

$$\varepsilon \quad \mapsto \quad 1z^0$$

SMC Model Counting

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

$$
\begin{aligned}
\varepsilon &\mapsto 1z^0 \\
c &\mapsto 1z^1
\end{aligned}
$$

SMC Model Counting

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

$$
\begin{array}{lcl}
\varepsilon & \mapsto & 1z^0 \\
c & \mapsto & 1z^1 \\
A|B & \mapsto & A(z) + B(z)
\end{array}
$$

SMC Model Counting

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

$$
\begin{array}{lcl}
\varepsilon & \mapsto & 1z^0 \\
c & \mapsto & 1z^1 \\
A|B & \mapsto & A(z) + B(z) \\
A \circ B & \mapsto & A(z) \times B(z)
\end{array}
$$

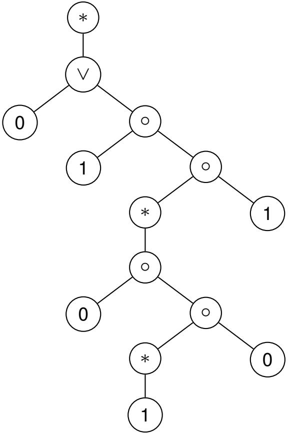
SMC Model Counting

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

$$
\begin{array}{lcl}
\varepsilon & \mapsto & 1z^0 \\
c & \mapsto & 1z^1 \\
A|B & \mapsto & A(z) + B(z) \\
A \circ B & \mapsto & A(z) \times B(z) \\
A^* & \mapsto & 1/(1 - A(z))
\end{array}
$$

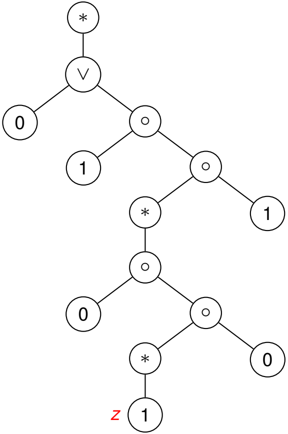
Regular Expressions

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



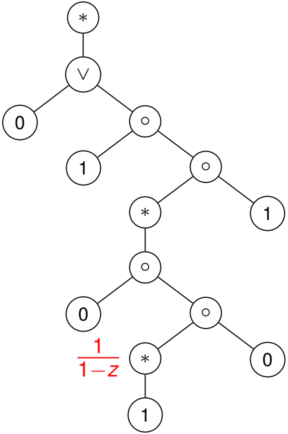
Regular Expressions

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



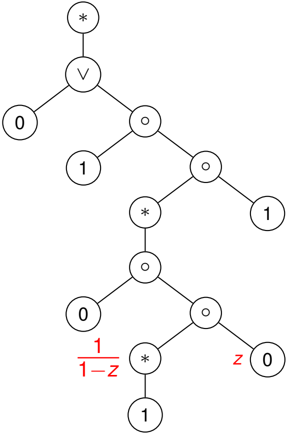
Regular Expressions

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



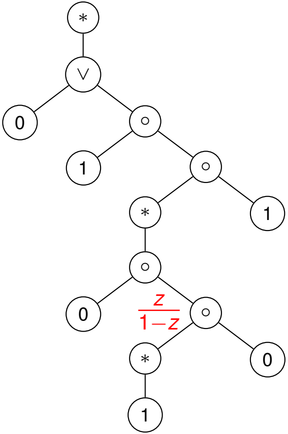
Regular Expressions

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



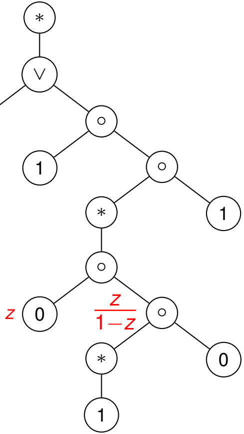
Regular Expressions

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



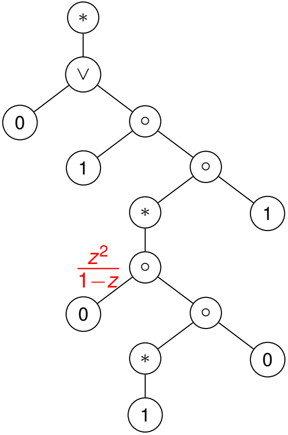
Regular Expressions

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



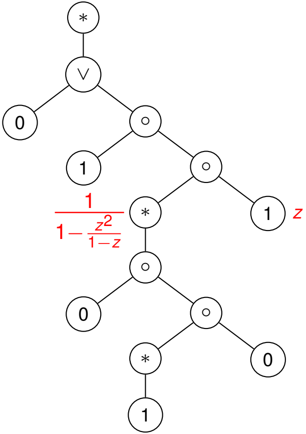
Regular Expressions

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



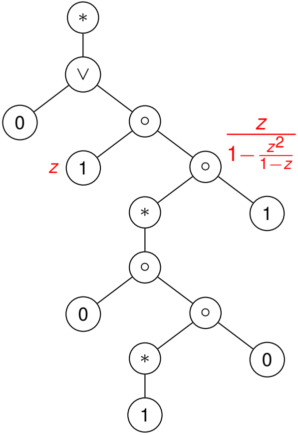
Regular Expressions

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



Regular Expressions

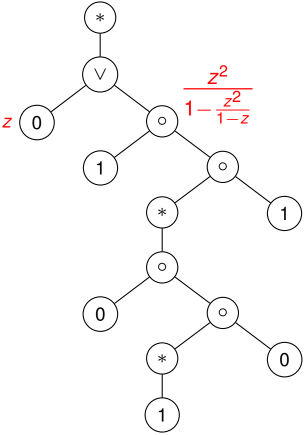
$$X \in (0|(1(01^*0)^*1))^*$$



$$\frac{z}{1-\frac{z^2}{1-z}}$$

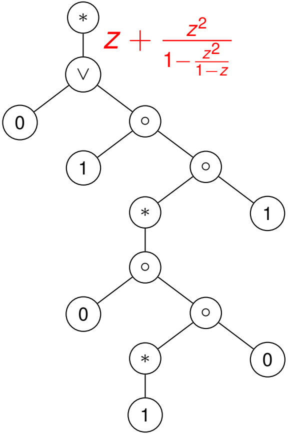
Regular Expressions

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



Regular Expressions

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

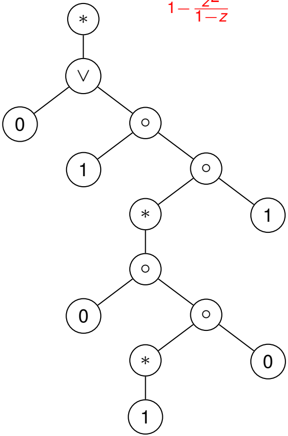


$$z + \frac{z^2}{1-\frac{z^2}{1-z}}$$

Regular Expressions

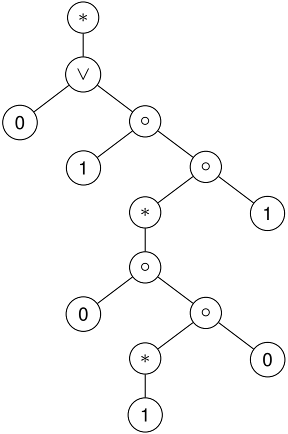
$$X \in (0|(1(01^*0)^*1))^*$$

$$\frac{1}{1-z-\frac{z^2}{1-\frac{z^2}{1-z}}}$$



Regular Expressions

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

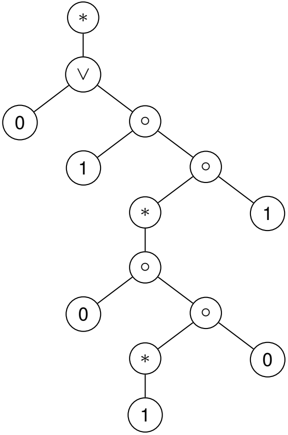


Generating Function:

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

Regular Expressions

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



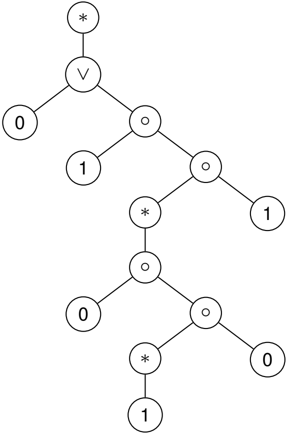
Generating Function:

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

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

Regular Expressions

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



Generating Function:

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

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

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

Other operations in SMC

Specialized transformations for other operations

Other operations in SMC

Specialized transformations for other operations

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

Other operations in SMC

Specialized transformations for other operations

$$
\begin{aligned}
contains(s_1, s_2) &\mapsto \tfrac{z^n}{(1-Mz)(z^n+(1-Mz)c(z))} \\
F_1 \vee F_2 &\mapsto [\max(L_1(z), L_2(z)), \min(U_1(z) + U_2(z), G(z))]
\end{aligned}
$$

Other operations in SMC

Specialized transformations for other operations

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

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

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

	bound	SMC lower bound	SMC upper bound	ABC count
nullhttpd	500	3752	3760	3760
ghttpd	620	4880	4896	4896
csplit	629	4852	4921	4921
grep	629	4676	4763	4763
wc	629	4281	4284	4281
obscure	6	0	3	2

Experimental Comparison

JavaScript Benchmarks

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

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.

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.
- 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

ABC Bonus: Model Counting Linear Integer Arithmetic

What is this language?

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

ABC Bonus: Model Counting Linear Integer Arithmetic

What is this language?

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

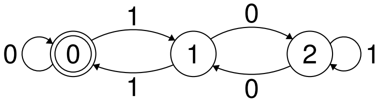
$$L(X) = \{s|s \text{ is a binary number divisible by 3}\}$$

ABC Bonus: Model Counting Linear Integer Arithmetic

What is this language?

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

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

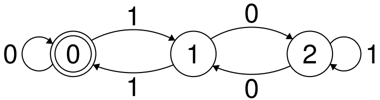


ABC Bonus: Model Counting Linear Integer Arithmetic

What is this language?

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

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



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

Model Counting Linear Integer Arithmetic

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

Model Counting Linear Integer Arithmetic

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

Constraints of the form:

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

Model Counting Linear Integer Arithmetic

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

Constraints of the form:

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

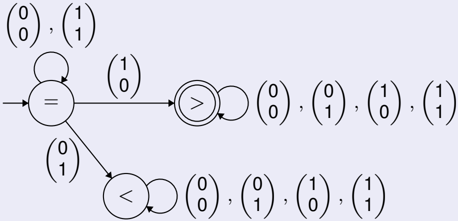
It is possible to represent the solutions to a set of LIA constraints as a binary multi-track DFA.

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 $x > y$.

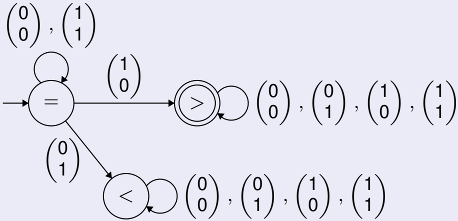


Binary Multi-track DFA

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 $x > y$.



Solutions of length $n \equiv$ solutions within bound 2^n

Model Counting Summary

Counting Techniques for Different Theories

- Boolean

Model Counting Summary

Counting Techniques for Different Theories

- Boolean
  - Truth Table (Brute Force)
  - DPLL

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

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

- Stanley. Enumerative Combinatorics Chapter 4. 2004.
- Sedgwick. Analytic Combinatorics Chapter 5: Generating Functions. 2009
- Biere. Handbook of Satisfiability. Chapter 20: Model Counting. 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
- Boigelot. Counting the solutions of Presburger equations without enumerating them. TCS 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 Barvinoks'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
- Phan. Model Counting Modulo Theories. PhD Thesis 2014.
- Birnbaum. The good old Davis-Putnam procedure helps counting models. JAIR 1999

Thank you.