What is Decidable about String Constraints with the ReplaceAll Function

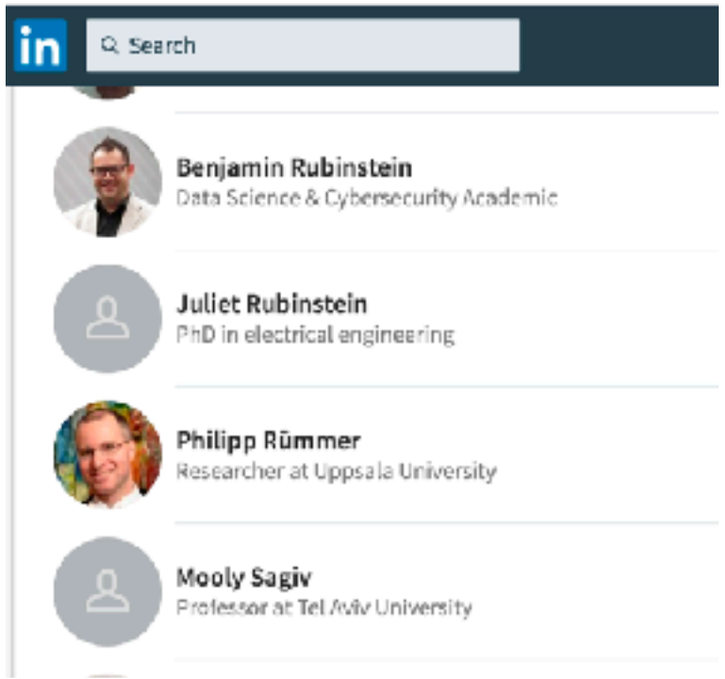
Taolue Chen (Birkbeck)
Yan Chen (Chinese Academy of Sciences)
Matthew Hague (Royal Holloway)
Anthony W. Lin (Oxford)
Zhilin Wu (Chinese Academy of Sciences)

String Data Type

Prevalent in today's software

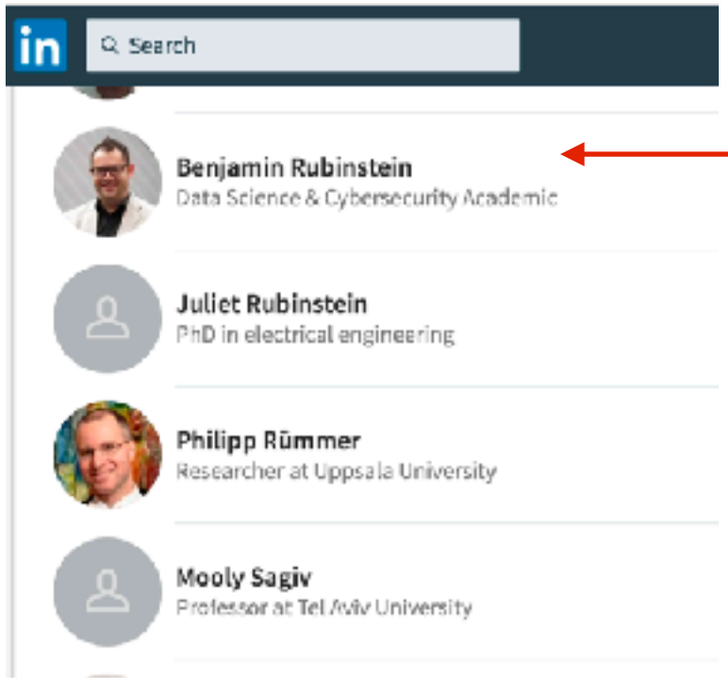
String Data Type

Prevalent in today's software



String Data Type

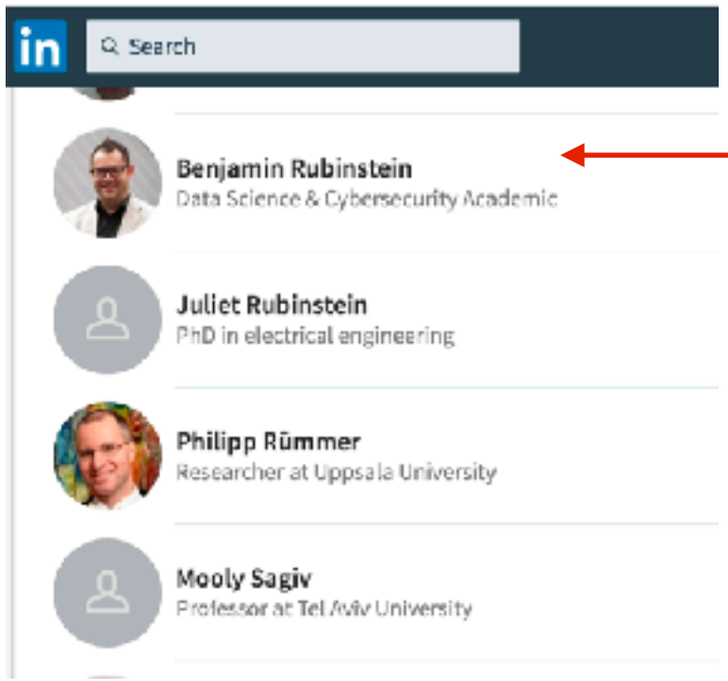
Prevalent in today's software



Ben

String Data Type

Prevalent in today's software



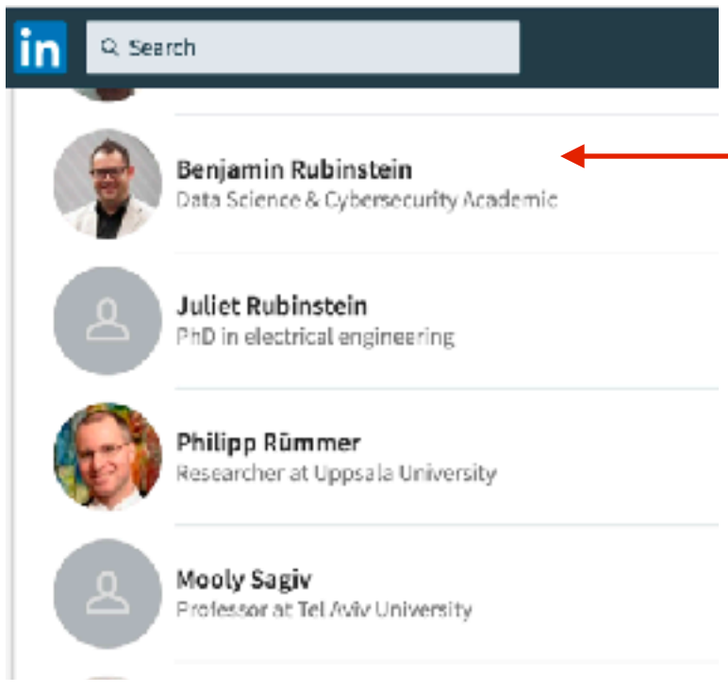
Ben

Dynamically generated by

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

String Data Type

Prevalent in today's software



Ben

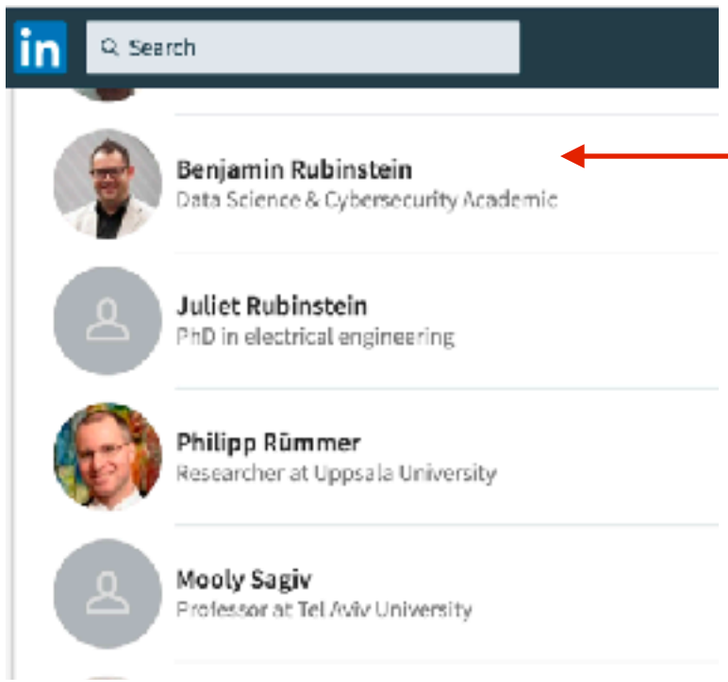
Dynamically generated by

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

Many string-related bugs — hard to find by random testing

String Data Type

Prevalent in today's software



Ben

Dynamically generated by

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

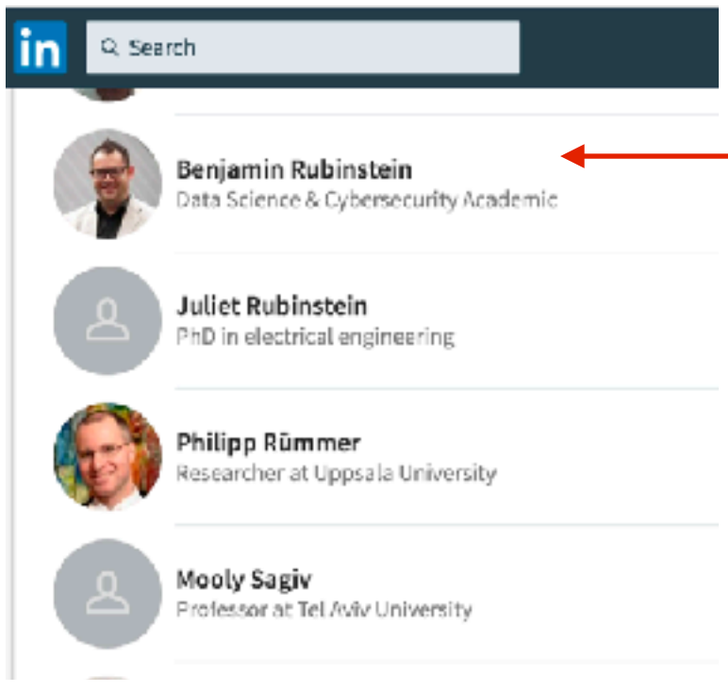
Many string-related bugs — hard to find by random testing

 ……

XSS

String Data Type

Prevalent in today's software



Ben

Dynamically generated by

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

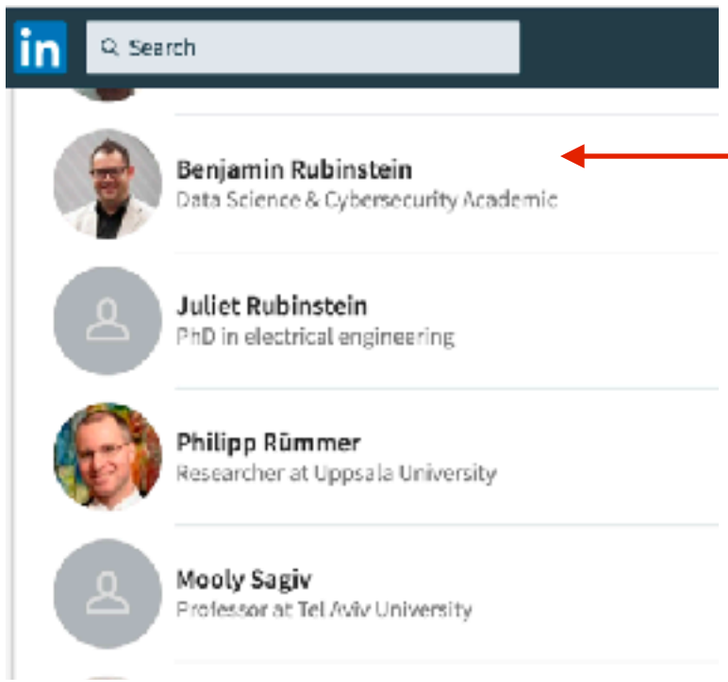
Many string-related bugs — hard to find by random testing

 ……

XSS

String Data Type

Prevalent in today's software



Ben

Dynamically generated by

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

Many string-related bugs — hard to find by random testing

 ……

XSS

Q: Does the sanitisation work?

String theory (a la SMT)

Constants/Variables: over the string domain (over a finite alphabet)

String operations:
- equality (=)
- concatenation (+)
- regex matching
- length function (len)
- replaceAll
- …

Formulas: quantifier-free, first-order

Problem: satisfiability (existence of a solution)

String theory (a la SMT)

Constants/Variables: over the string domain (over a finite alphabet)

String operations:
- equality (=)
- concatenation (+)
- regex matching
- length function (len)
- replaceAll
- …

Formulas: quantifier-free, first-order

Problem: satisfiability (existence of a solution)

(y + ‘ba’ + x) = (x + ‘ab’ + y)

String theory (a la SMT)

Constants/Variables: over the string domain (over a finite alphabet)

String operations:
- equality (=)
- concatenation (+)
- regex matching
- length function (len)
- replaceAll
- …

Formulas: quantifier-free, first-order

Problem: satisfiability (existence of a solution)

(y + ‘ba’ + x) = (x + ‘ab’ + y)

satisfiable: x -> ‘b’, y -> ‘’

String Solvers Everywhere

Kaluza	Z3	Z3-str
Kudzu	PISA	IBM AppScan
HAMPI	Saner	Sloth
S3	Stranger	STP
Norn	StrSolve	…
CVC4	SUSHI	

String Solvers Everywhere

Kaluza	Z3	Z3-str
Kudzu	PISA	IBM AppScan
HAMPI	Saner	Sloth
S3	Stranger	STP
Norn	StrSolve	…
CVC4	SUSHI	**Focus on "heuristics"**

Decidable String Theories

Decidable String Theories

Word Equations

(y + ‘ba’ + x = x + ‘ab’ + y)

Decidable [Makanin’77]

Decidable String Theories

Word Equations

(y + ‘ba’ + x = x + ‘ab’ + y)

Decidable [Makanin’77]

Word Equations with Regular Constraints

(y+‘ba’+x = x+‘ab’+y) /\ x in a*

Decidable [Schulz’90]

Decidable String Theories

Word Equations

(y + ‘ba’ + x = x + ‘ab’ + y)

Decidable [Makanin’77]

Word Equations with Regular Constraints

(y+‘ba’+x = x+‘ab’+y) /\ x in a*

Decidable [Schulz’90]

Theory of Concatenation with Regular Constraints

s2 = s1+s1 /\ s3+s2 !=s1+s7+s8
/\ s1 in a* /\ s3 in b*a*

Decidable [Buchi&Senger’90]

Decidable String Theories

Word Equations

(y + ‘ba’ + x = x + ‘ab’ + y)

Decidable [Makanin’77]

Word Equations with Regular Constraints

(y+‘ba’+x = x+‘ab’+y) /\ x in a*

Decidable [Schulz’90]

Theory of Concatenation with Regular Constraints

s2 = s1+s1 /\ s3+s2 !=s1+s7+s8
/\ s1 in a* /\ s3 in b*a*

Decidable [Buchi&Senger’90]

Word Equations with Length Constraints

(y + ‘ba’ + x = x + ‘ab’ + y) /\
(len(x) = len(y))

Long-standing classical open problem

Decidable String Theories

Word Equations

(y + ‘ba’ + x = x + ‘ab’ + y)

Decidable [Makanin’77]

Word Equations with Regular Constraints

(y+‘ba’+x = x+‘ab’+y) /\ x in a*

Decidable [Schulz’90]

Theory of Concatenation with Regular Constraints

s2 = s1+s1 /\ s3+s2 !=s1+s7+s8
/\ s1 in a* /\ s3 in b*a*

Decidable [Buchi&Senger’90]

Word Equations with Length Constraints

(y + ‘ba’ + x = x + ‘ab’ + y) /\
(len(x) = len(y))

Long-standing classical open problem

Many string operations are still missing

Problem: **replaceAll** is by and large missing

Problem: **replaceAll** is by and large missing

Proposal: add **replaceAll** to string theories in a decidable way

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

Output: *subject* with *all* occurrences of strings matching *pat* replaced by *rep*

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

Output: *subject* with *all* occurrences of strings matching *pat* replaced by *rep*

In VIM: %s/*pat*/*rep*/g

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

Output: *subject* with *all* occurrences of strings matching *pat* replaced by *rep*

In VIM: %s/*pat*/*rep*/g

The Road Not Taken

BY ROBERT FROST
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

subject

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

Output: *subject* with *all* occurrences of strings matching *pat* replaced by *rep*

In VIM: %s/*pat*/*rep*/g

The Road Not Taken

BY ROBERT FROST
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

subject

pat = “Two”

rep = “Three”

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

Output: *subject* with *all* occurrences of strings matching *pat* replaced by *rep*

In VIM: %s/*pat*/*rep*/g

The Road Not Taken

BY ROBERT FROST
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

subject

pat = “Two”

rep = “Three”

The ReplaceAll Function

replaceAll(*subject*,*pat*,*rep*)

Output: *subject* with *all* occurrences of strings matching *pat* replaced by *rep*

In VIM: %s/*pat*/*rep*/g

The Road Not Taken

BY ROBERT FROST
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

subject

pat = “Two”

rep = “Three”

The Road Not Taken

BY ROBERT FROST
Three roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Three roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

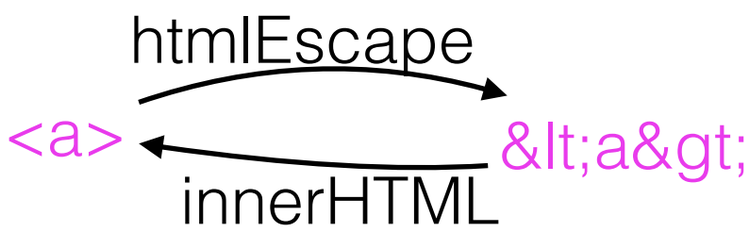
%s/Two/Three/g

Application I: Sanitisers

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

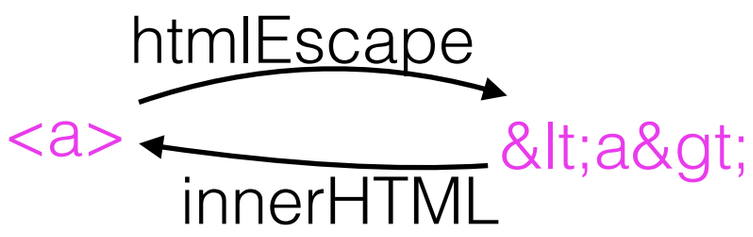
Application I: Sanitisers

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```



Application I: Sanitisers

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```



Application II: Web Templating

Application II: Web Templating

HTML template (with Mustache)

```
...
<h1> User <span
 onclick=“popupText('{{bio}}')">
   {{userName}}</span> </h1>
...
```

Application II: Web Templating

HTML template (with Mustache)

```
...
<h1> User <span
 onclick="popupText('{{bio}}')">
   {{userName}}</span> </h1>
...
```

JSON files

```
...
bio = "John is 19";
userName = "John";
...
```

Application II: Web Templating

HTML template (with Mustache)

```
...
<h1> User <span
 onclick="popupText('{{bio}}')">
   {{userName}}</span> </h1>
...
```

JSON files

```
...
bio = "John is 19";
userName = "John";
...
```

HTML

```
...
<h1> User <span
 onclick="popupText('John is 19')">
   John</span> </h1>
...
```

Application II: Web Templating

HTML template (with Mustache)

```
...
<h1> User <span
 onclick="popupText('{{bio}}')">
   {{userName}}</span> </h1>
...
```

JSON files

```
...
bio = "'); attackScript('";
userName = "Evil";
...
```

HTML

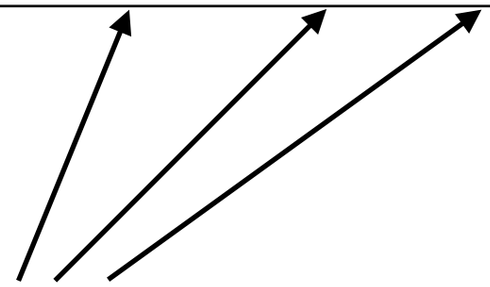
```
...
<h1> User <span
 onclick="popupText('); attackScript('')">
   Evil</span> </h1>
...
```

replaceAll in String Theory

x = **replaceAll**(*subject*,*pat*,*rep*)

replaceAll in String Theory

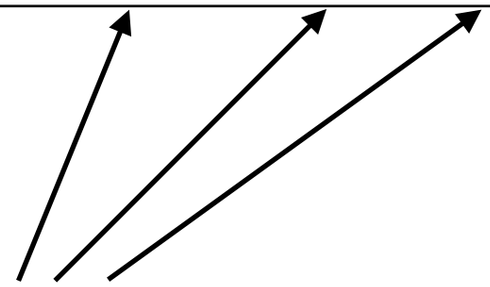
x = **replaceAll**(*subject*,*pat*,*rep*)



Can be a string constant/variable

replaceAll in String Theory

x = **replaceAll**(*subject*,*pat*,*rep*)



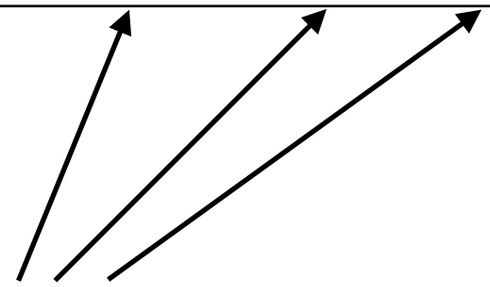
Can be a string constant/variable

pat can be a regular expression (over string constants)

(semantics: leftmost/longest match)

replaceAll in String Theory

x = **replaceAll**(*subject*,*pat*,*rep*)



Can be a string <u>constant</u>/<u>variable</u>

pat can be a <u>regular expression (over string constants)</u>

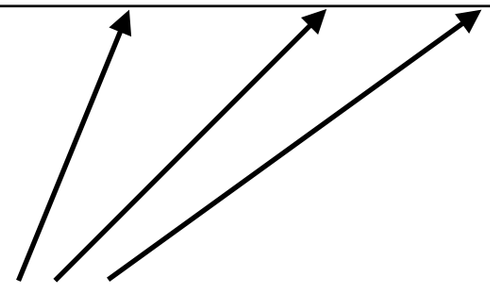
(semantics: leftmost/longest match)

Most common usage: pat/rep are constants

escapeString(x,z) := y = replaceAll(x,"",\") /\ z = replaceAll(y,',\')

replaceAll in String Theory

x = **replaceAll**(*subject*,*pat*,*rep*)



Can be a string constant/variable

pat can be a regular expression (over string constants)

(semantics: leftmost/longest match)

Most common usage: pat/rep are constants

```
escapeString(x,z) :=      y = replaceAll(x,'',\'') /\ z = replaceAll(y,',\')
```

Not so uncommon usage: rep is a variable, pat is a constant

```
mustache(x,z,bio,userName) :=      y = replaceAll(x,'{{bio}}',bio) /\
                                   z = replaceAll(y,'{{userName}}',userName)
```

Bad News

Proposition (Folklore): String constraints with equality, regex, and replaceAll (pat/rep constants) is **undecidable**

Bad News

Proposition (Folklore): String constraints with equality, regex, and replaceAll (pat/rep constants) is **undecidable**

Easy reduction from Post Correspondence Problem

Bad News

Proposition (Folklore): String constraints with equality, regex, and replaceAll (pat/rep constants) is **undecidable**

Easy reduction from Post Correspondence Problem

1	2	3
a	ab	bba
baa	aa	bb

Bad News

Proposition (Folklore): String constraints with equality, regex, and replaceAll (pat/rep constants) is **undecidable**

Easy reduction from Post Correspondence Problem

1	2	3
a	ab	bba
baa	aa	bb

x in (1+2+3)* /\
y = replaceAll(x,1,a) /\ y' = replaceAll(y,2,ab) /\ y'' = replaceAll(y',3,bba) /\
z = replaceAll(x,1,baa) /\ z' = replaceAll(z,2,aa) /\ z'' = replaceAll(z,3,bb) /\
y'' = z''

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/finite transducer
2. assert.: regular constraint

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

includes: replaceAll(*VAR***,**const**,**const**)**

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/finite transducer
2. assert.: regular constraint

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

includes: replaceAll(*VAR*,const,const)

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/finite transducer
2. assert.: regular constraint

Key Idea: NO general string equality in conditionals!

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

```
x := x + ‘aba’ + y;
y := replaceall(x,’a’,’c’);
assert( y in (‘b’)* )
```

Program

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

```
x := x + ‘aba’ + y;
y := replaceall(x,’a’,’c’);
assert( y in (‘b’)* )
```

Program

```
x1 := x + ‘aba’ + y ∧
y1 := replaceall(x1,’a’,’c’) ∧
assert( y1 in (‘b’)* )
```

Formula (use SSA form)

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

$$S ::= \quad y := f(x_1, \ldots, x_n) \mid \mathbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2$$

where

$$f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\}$$

Path Feasibility Problem: decide if there exist input strings that satisfy all the assertions

```
x := x + ‘aba’ + y;
y := replaceall(x,’a’,’c’);
assert( y in (‘b’)* )
```

Program

```
x1 := x + ‘aba’ + y ∧
y1 := replaceall(x1,’a’,’c’) ∧
assert( y1 in (‘b’)* )
```

Formula (use SSA form)

Path Feasibility = Satisfiability (in disguise)

Why Straight-Line

Why Straight-Line

- Prohibits bad formulas

Why Straight-Line

- Prohibits bad formulas

x in (1+2+3)* /\
y = replaceAll(x,1,a) /\ y' = replaceAll(y,2,ab) /\ y'' = replaceAll(y',3,bba) /\
z = replaceAll(x,1,baa) /\ z' = replaceAll(z,2,aa) /\ z'' = replaceAll(z,3,bb) /\
y'' = z''

Why Straight-Line

- Prohibits bad formulas

x in (1+2+3)* /\
y = replaceAll(x,1,a) /\ y' = replaceAll(y,2,ab) /\ y'' = replaceAll(y',3,bba) /\
z = replaceAll(x,1,baa) /\ z' = replaceAll(z,2,aa) /\ z'' = replaceAll(z,3,bb) /\
y'' = z''

Why Straight-Line

- Prohibits bad formulas

x in (1+2+3)* /\
y = replaceAll(x,1,a) /\ y' = replaceAll(y,2,ab) /\ y'' = replaceAll(y',3,bba) /\
z = replaceAll(x,1,baa) /\ z' = replaceAll(z,2,aa) /\ z'' = replaceAll(z,3,bb) /\
y'' = z''

- Captures constraints from symbolic execution in practice

Why Straight-Line

- Prohibits bad formulas

x in (1+2+3)* /\
y = replaceAll(x,1,a) /\ y' = replaceAll(y,2,ab) /\ y'' = replaceAll(y',3,bba) /\
z = replaceAll(x,1,baa) /\ z' = replaceAll(z,2,aa) /\ z'' = replaceAll(z,3,bb) /\
y'' = z''

- Captures constraints from symbolic execution in practice

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

assert(nameElem matches
‘ ……)

Why Straight-Line

- Prohibits bad formulas

x in (1+2+3)* /\
y = replaceAll(x,1,a) /\ y' = replaceAll(y,2,ab) /\ y'' = replaceAll(y',3,bba) /\
z = replaceAll(x,1,baa) /\ z' = replaceAll(z,2,aa) /\ z'' = replaceAll(z,3,bb) /\
y'' = z''

- Captures constraints from symbolic execution in practice

```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

assert(nameElem matches
‘ ……)

LOTS of existing benchmarks are in SL

Limitation of SL [LB'16]

HTML template (with Mustache)

```
...
<h1> User <span
 onclick="popupText('{{bio}}')">
   {{userName}}</span> </h1>
...
```

JSON files

```
...
bio = "'); attackScript('";
userName = "Evil";
...
```

HTML

```
...
<h1> User <span
 onclick="popupText('); attackScript('')">
   Evil</span> </h1>
...
```

Limitation of SL [LB'16]

HTML template (with Mustache)

```
…
<h1> User <span
 onclick="popupText('{{bio}}')">
   {{userName}}</span> </h1>
…
```

JSON files

```
…
bio = "'); attackScript('";
userName = "Evil";
…
```

HTML

```
…
<h1> User <span
 onclick="popupText('); attackScript('')">
   Evil</span> </h1>
…
```

Requires more general replaceall!

x = replaceAll(text,'{{bio}}',bio)

Our Main Result

A more expressive decidable straight-line fragment with **replaceAll**!

Our Main Result

A more expressive decidable straight-line fragment with **replaceAll**!

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/**replaceAll**(VAR,regex,VAR)
2. assertion: regular constraint

Our Main Result

A more expressive decidable straight-line fragment with **replaceAll**!

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/**replaceAll**(VAR,regex,VAR)
2. assertion: regular constraint

Can model string operations used in auto-sanitisation in web templates!

Closure, Angular, Handlebars

Our Main Result

A more expressive decidable straight-line fragment with **replaceAll**!

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/**replaceAll**(VAR,regex,VAR)
2. assertion: regular constraint

Can model string operations used in auto-sanitisation in web templates!

Closure, Angular, Handlebars

This decidability was surprising since arithmetic can be simulated!

Our Main Result

A more expressive decidable straight-line fragment with **replaceAll**!

Theorem: Path feasibility is decidable whenever:

1. assign.: concatenation/**replaceAll**(VAR,regex,VAR)
2. assertion: regular constraint

Can model string operations used in auto-sanitisation in web templates!

Closure, Angular, Handlebars

This decidability was surprising since arithmetic can be simulated!

5 ———> ‘aaaaa’

z = x*y ———> z = **replaceAll**(x,’a’,y)

z = x+y ———> z = x+y (concatenation)

Proof Idea

Lemma: Concatenation in SL can be expressed as **replaceAll**(VAR,regex,VAR).

Proof Idea

Lemma: Concatenation in SL can be expressed as **replaceAll**(VAR,regex,VAR).

X = Y + ‘aba’ + Y + Z

Proof Idea

Lemma: Concatenation in SL can be expressed as **replaceAll**(VAR,regex,VAR).

X = Y + ‘aba’ + Y + Z ⟶

X0 = replaceAll(‘yabayz’,y,Y) /\
X = replaceAll(X’,z,Z)

Proof Idea

Definition: The pre-image of a language *L* under **replaceAll**$_{\text{pat}}$ with pattern pat is:

$$\mathbf{replaceAll}^{-1}_{\text{pat}}(L) := \{(v, w) : \mathbf{replaceAll}(v, \text{pat}, w) \in L\}$$

L = ‘Hi [A-Z][a-z]*,[A-Z][a-z]* is a nice name’

pat = ‘9’

Complexity Consideration

Theorem: Our algorithm has the same complexity as LB'16 SL: EXPSPACE (double exp-time).

Best lower bound is PSPACE-hardness

We identified a PSPACE-complete fragment of our constraint language

Extensions

Length Constraints

assert(len(x) = len(y))

Length Constraints

assert(len(x) = len(y))

Theorem: Path feasibility is **un**decidable if:

1. assign.: applies **replaceAll**(VAR,const,VAR)
2. assert.: regular constraint or length constraints

Length Constraints

assert(len(x) = len(y))

Theorem: Path feasibility is **un**decidable if:

1. assign.: applies **replaceAll**(VAR,const,VAR)
2. assert.: regular constraint or length constraints

Can encode existence of solutions to polynomials (all over Nat. Numbers):

$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$$

Length Constraints

assert(len(x) = len(y))

Theorem: Path feasibility is **un**decidable if:

1. assign.: applies **replaceAll**(VAR,const,VAR)
2. assert.: regular constraint or length constraints

Can encode existence of solutions to polynomials (all over Nat. Numbers):

$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$$

Note:

1. Some length constraints are regular

   len(x) < 7 len(x) mod 7 = 3

2. [LB'16] decidable for **replaceAll**(VAR,regex,const)

Variable in the Pattern

Proof is by a reduction from PCP

Variable in the Pattern

Theorem: Path feasibility is **un**decidable whenever:

1. assign.: **replaceAll**(VAR,VAR,const)
2. assert.: regular constraint

Proof is by a reduction from PCP

Final Words

Summary:

- Decidability boundary of string solving with **replaceAll**
- Reason to be positive!

Ongoing work:

- Computational complexity issues
- Unify transducers [LB’16] with replaceAll(VAR,const,VAR)
- String solver based on our constraint language