What is Decidable about String Constraints with the ReplaceAll Function

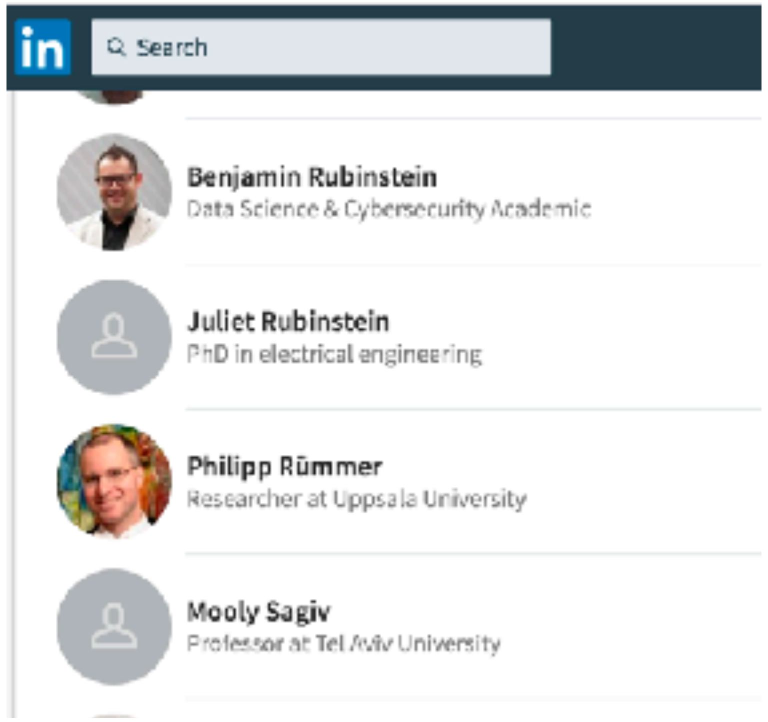
Taolue Chen (Birkbeck)
Yan Chen (Chinese Academy of Sciences)
Matthew Hague (Royal Holloway)
Anthony W. Lin (Oxford)
Zhilin Wu (Chinese Academy of Sciences)

String Data Type

Prevalent in today's software

String Data Type

Prevalent in today's software



Benjamin Rubinstein
Data Science & Cybersecurity Academic

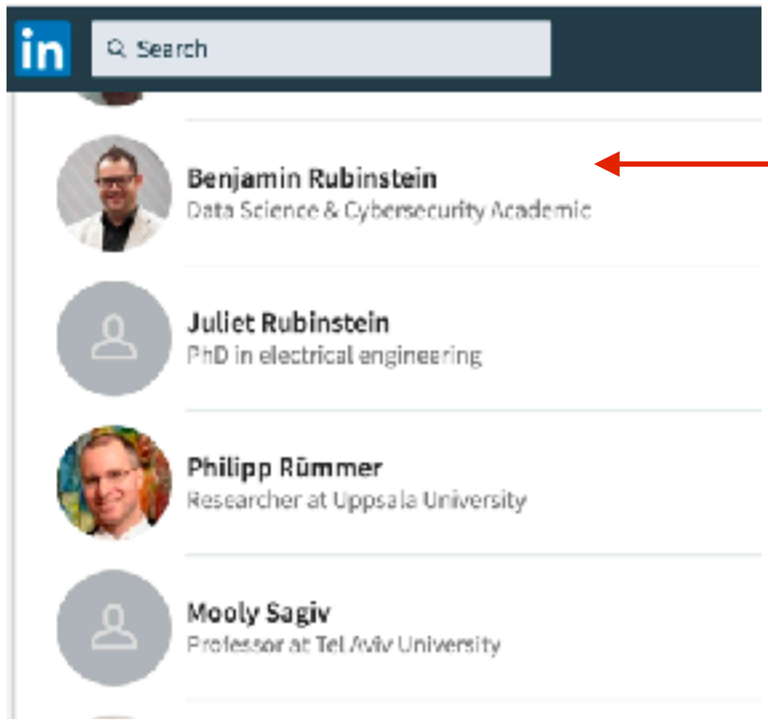
Juliet Rubinstein
PhD in electrical engineering

Philipp Rümmer
Researcher at Uppsala University

Mooly Sagiv
Professor at Tel Aviv University

String Data Type

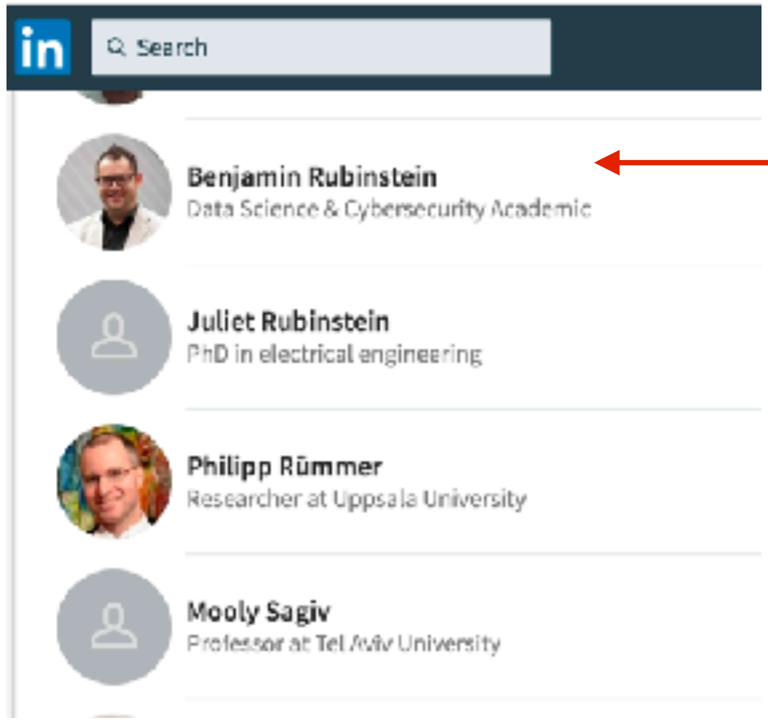
Prevalent in today's software


in
Search
Benjamin Rubinstein
Data Science & Cybersecurity Academic
Juliet Rubinstein
PhD in electrical engineering
Philipp Rümmer
Researcher at Uppsala University
Mooly Sagiv
Professor at Tel Aviv University

```
<a onclick="viewPerson('Ben')">Ben</a>
```

String Data Type

Prevalent in today's software



Benjamin Rubinstein
Data Science & Cybersecurity Academic

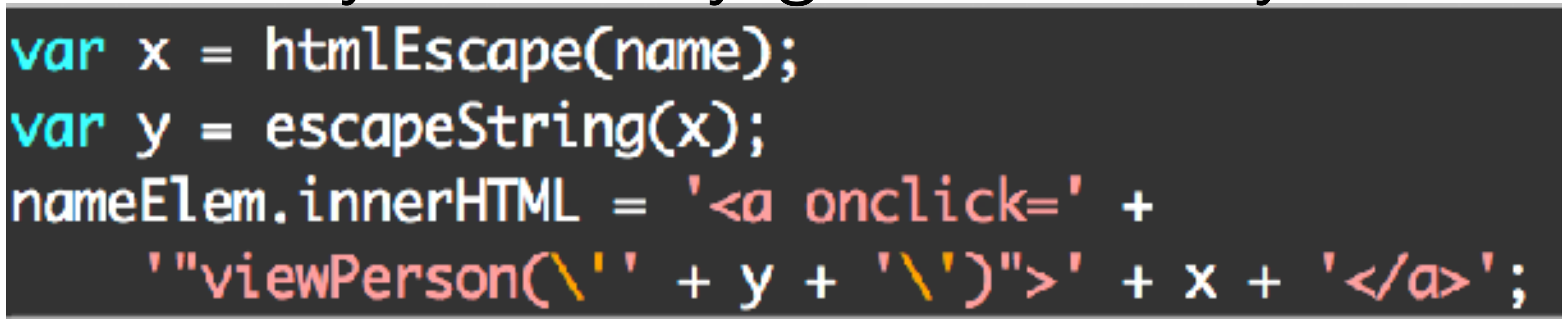
Juliet Rubinstein
PhD in electrical engineering

Philipp Rümmer
Researcher at Uppsala University

Mooly Sagiv
Professor at Tel Aviv University

<a onclick="viewPerson('Ben')">Ben</a>

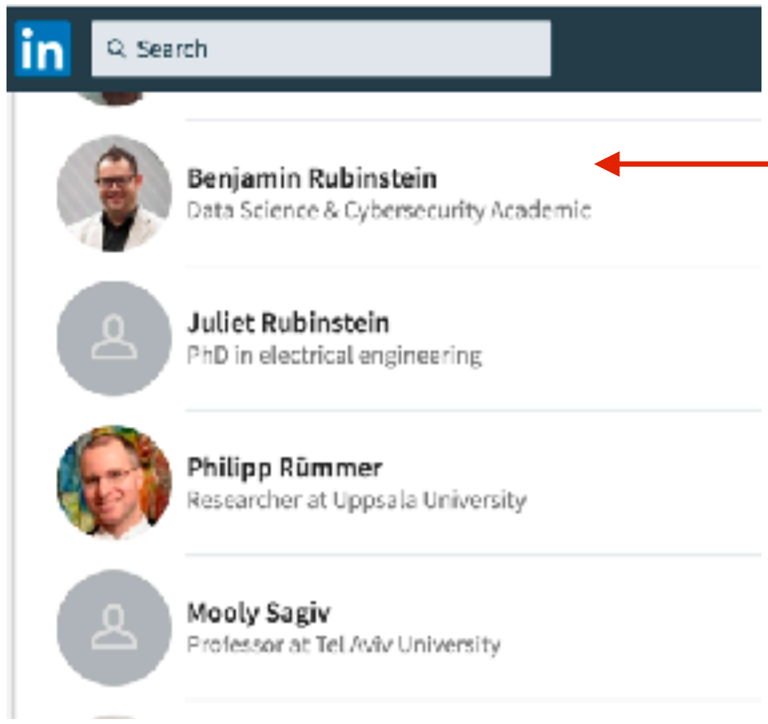
Dynamically generated by



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

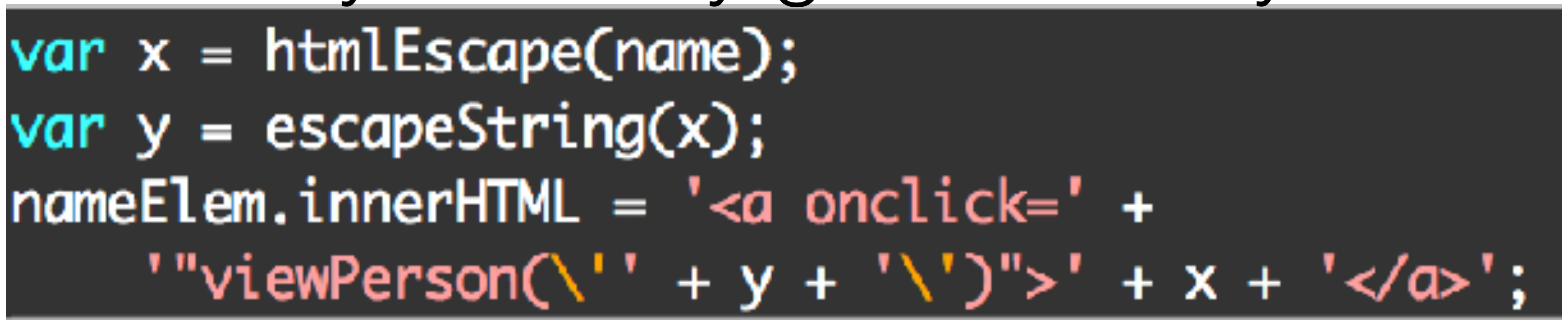
String Data Type

Prevalent in today's software


Search
Benjamin Rubinstein
Data Science & Cybersecurity Academic
Juliet Rubinstein
PhD in electrical engineering
Philipp Rümmer
Researcher at Uppsala University
Mooly Sagiv
Professor at Tel Aviv University

<a onclick="viewPerson('Ben')">Ben</a>

Dynamically generated by

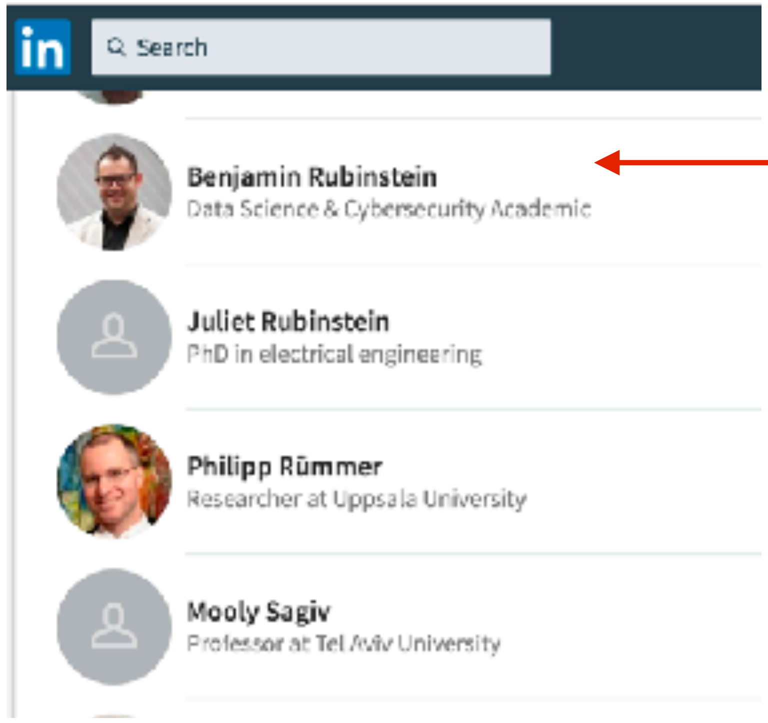


```
var x = htmlEscape(name);
var y = escapeString(x);
nameElem.innerHTML = '<a onclick=' +
    '"viewPerson(\'' + y + '\')">' + x + '</a>';
```

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

String Data Type

Prevalent in today's software


Benjamin Rubinstein
Data Science & Cybersecurity Academic

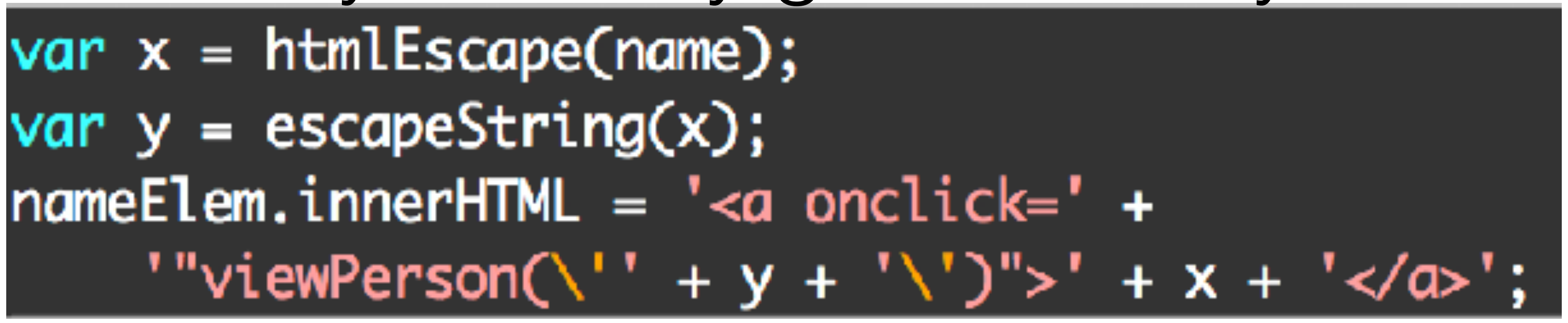
Juliet Rubinstein
PhD in electrical engineering

Philipp Rümmer
Researcher at Uppsala University

Mooly Sagiv
Professor at Tel Aviv University

<a onclick="viewPerson('Ben')">Ben</a>

Dynamically generated by



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

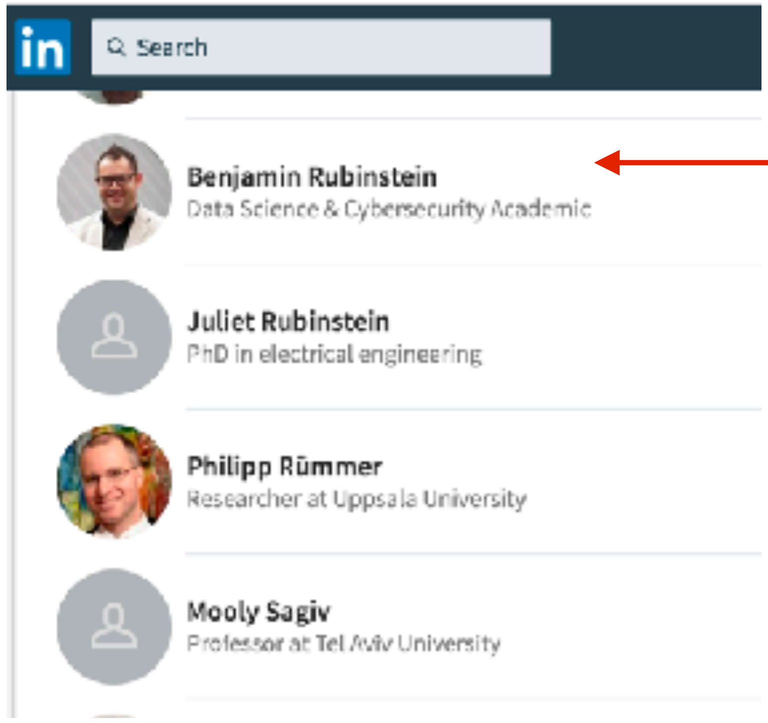
Many string-related bugs — hard to find by random testing

<a onclick="viewPerson(''); attackScript();......"> ...... </a>

XSS

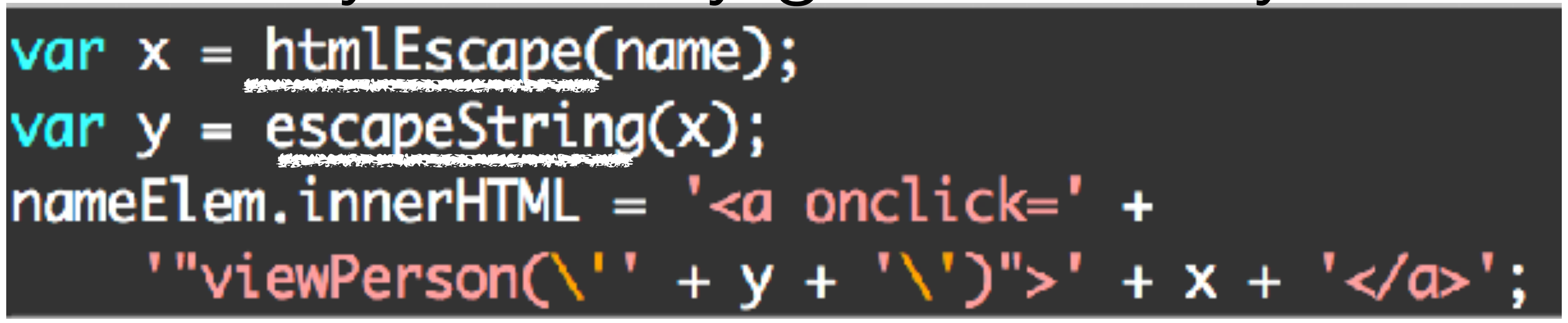
String Data Type

Prevalent in today's software


Search
Benjamin Rubinstein
Data Science & Cybersecurity Academic
Juliet Rubinstein
PhD in electrical engineering
Philipp Rümmer
Researcher at Uppsala University
Mooly Sagiv
Professor at Tel Aviv University

<a onclick="viewPerson('Ben')">Ben</a>

Dynamically generated by



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

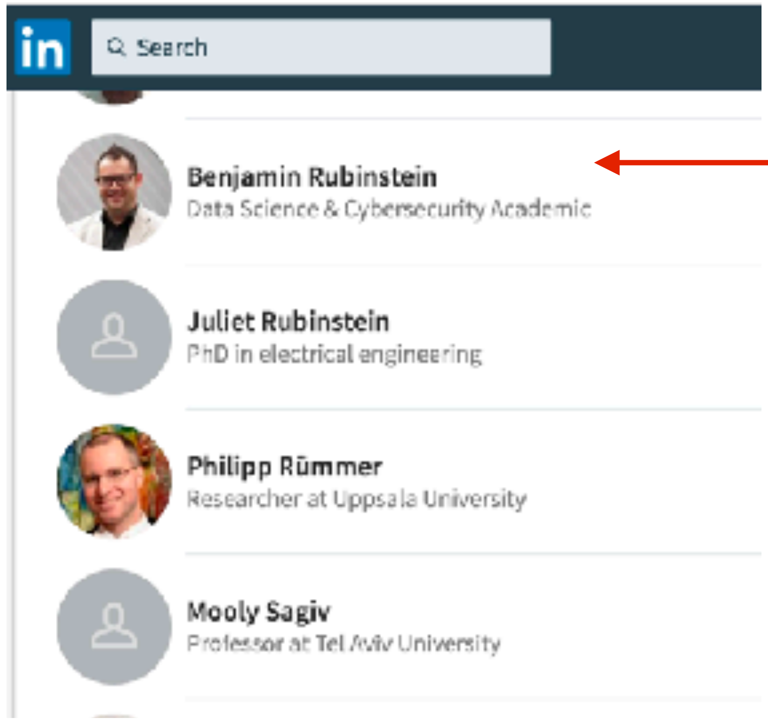
Many string-related bugs — hard to find by random testing

<a onclick="viewPerson(''); attackScript();……"> …… </a>

XSS

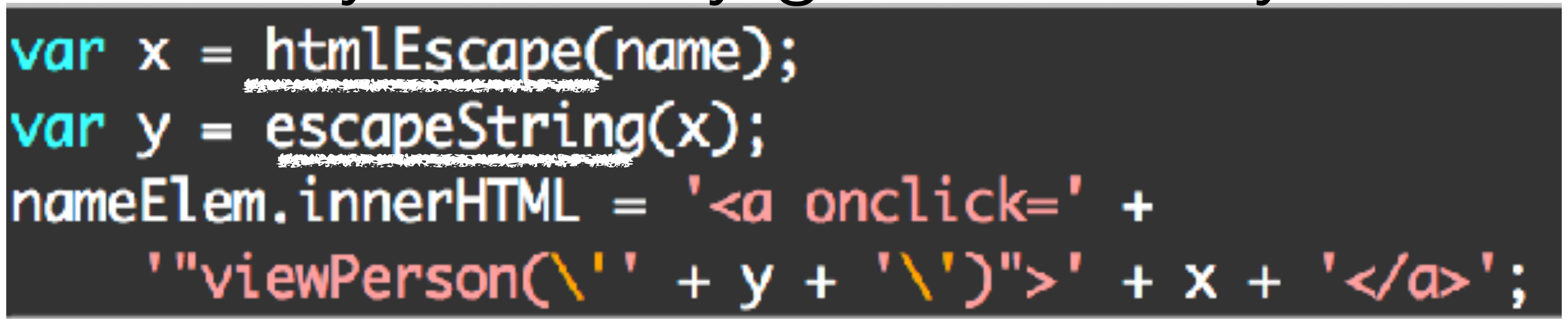
String Data Type

Prevalent in today's software


Search
Benjamin Rubinstein
Data Science & Cybersecurity Academic
Juliet Rubinstein
PhD in electrical engineering
Philipp Rümmer
Researcher at Uppsala University
Mooly Sagiv
Professor at Tel Aviv University

<a onclick="viewPerson('Ben')">Ben</a>

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

<a onclick="viewPerson(''); attackScript();……"> …… </a>

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

$\boxed{(y + \text{‘ba’} + x = x + \text{‘ab’} + y)}$

Decidable [Makanin’77]

Decidable String Theories

Word Equations

$\boxed{(y + \text{‘ba’} + x = x + \text{‘ab’} + y)}$ Decidable [Makanin’77]

Word Equations with Regular Constraints

$\boxed{(y+\text{‘ba’}+x = x+\text{‘ab’}+y) \wedge x \text{ in } a^*}$ 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

$\boxed{\textbf{replaceAll}(\textit{subject},\textit{pat},\textit{rep})}$

The ReplaceAll Function

$\boxed{\textbf{replaceAll}(\textit{subject},\textit{pat},\textit{rep})}$

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

The ReplaceAll Function

$\boxed{\textbf{replaceAll}(\textit{subject},\textit{pat},\textit{rep})}$

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

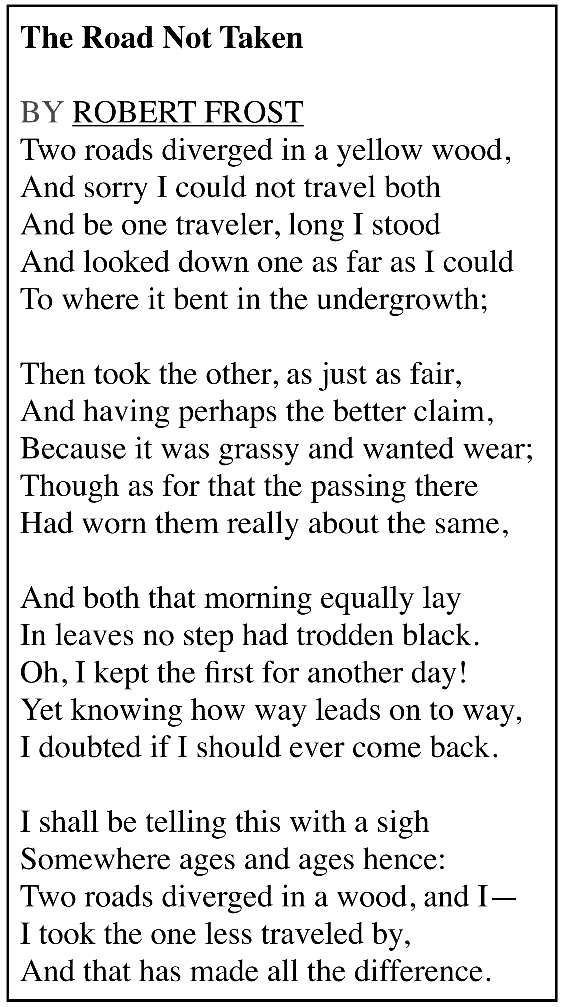
In VIM: %s/*pat*/*rep*/g

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.

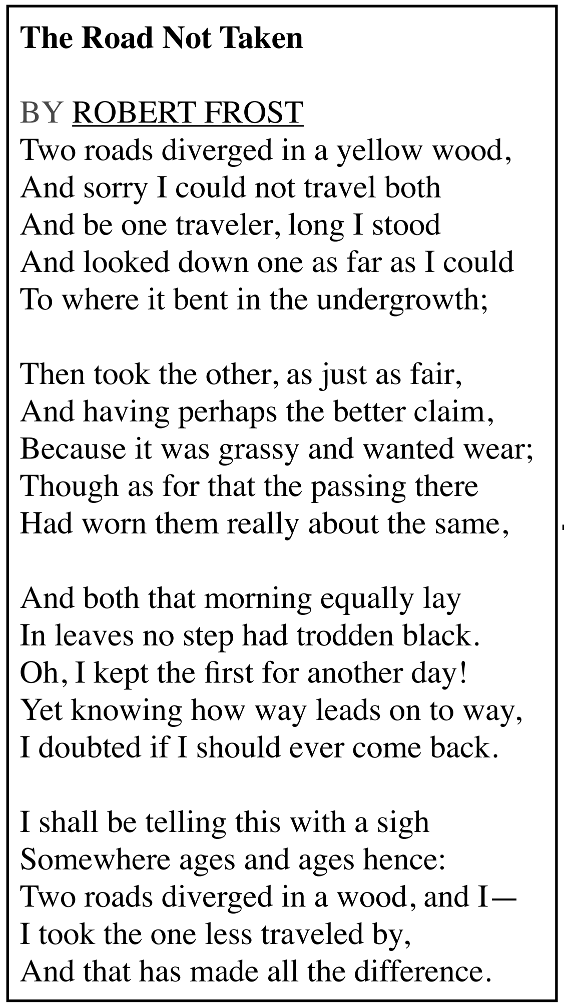
subject

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

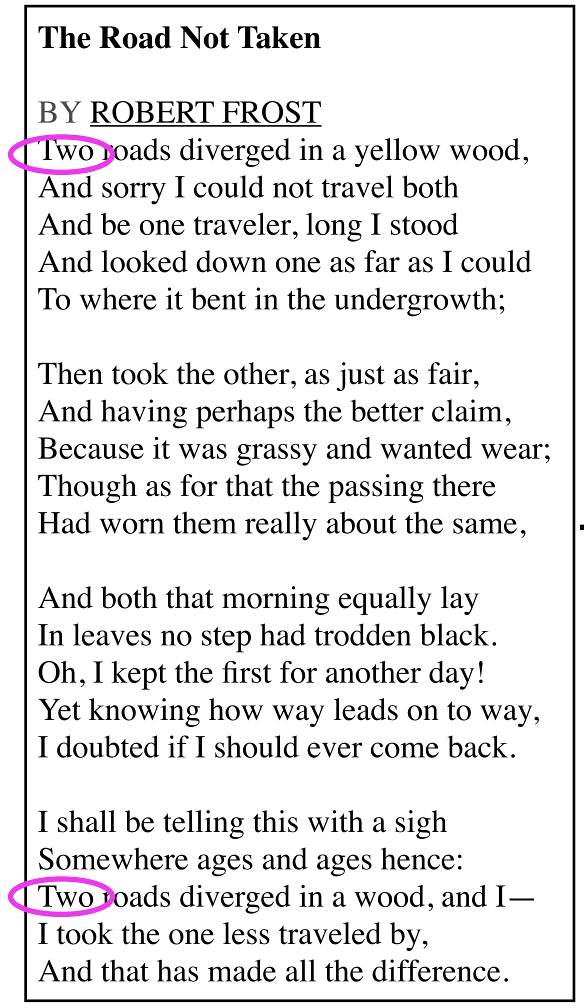
$$\xrightarrow[\textit{rep} = \text{“Three”}]{\textit{pat} = \text{“Two”}}$$

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”

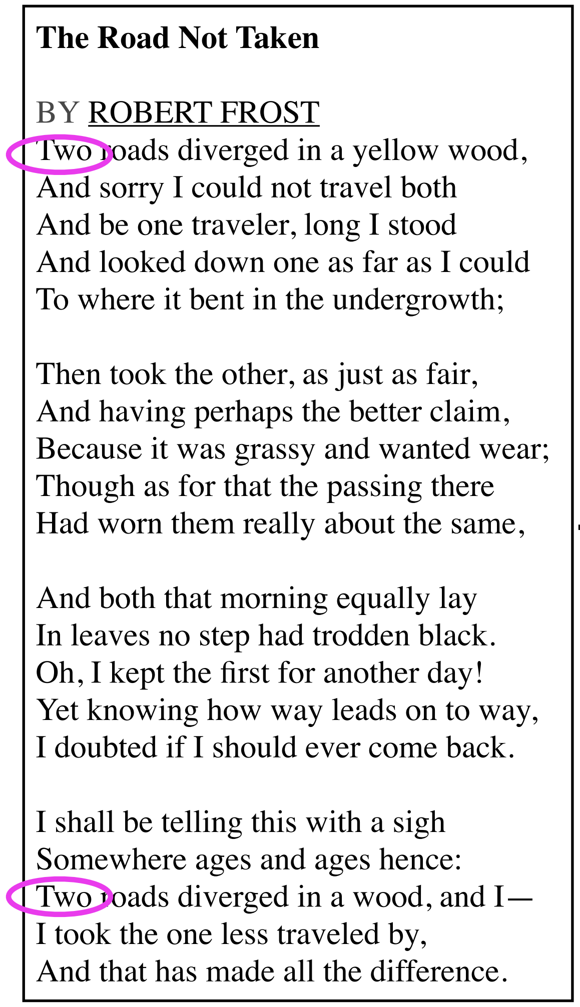
$\longrightarrow$

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,

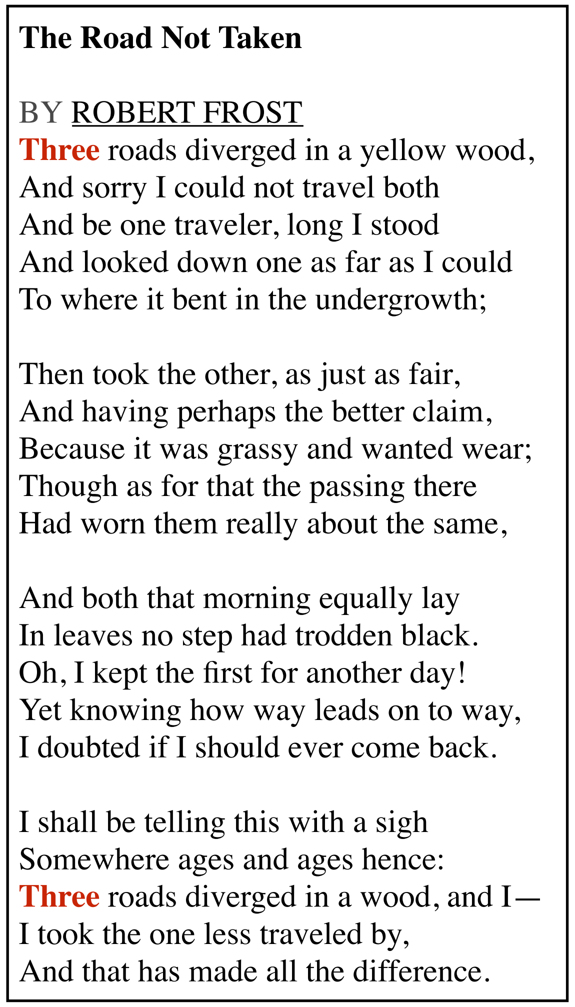
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

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;

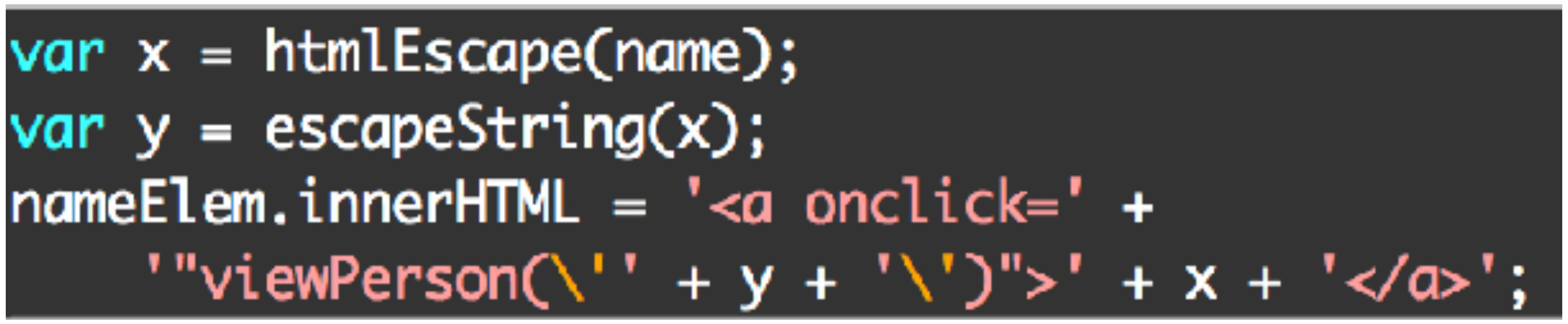
Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

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.

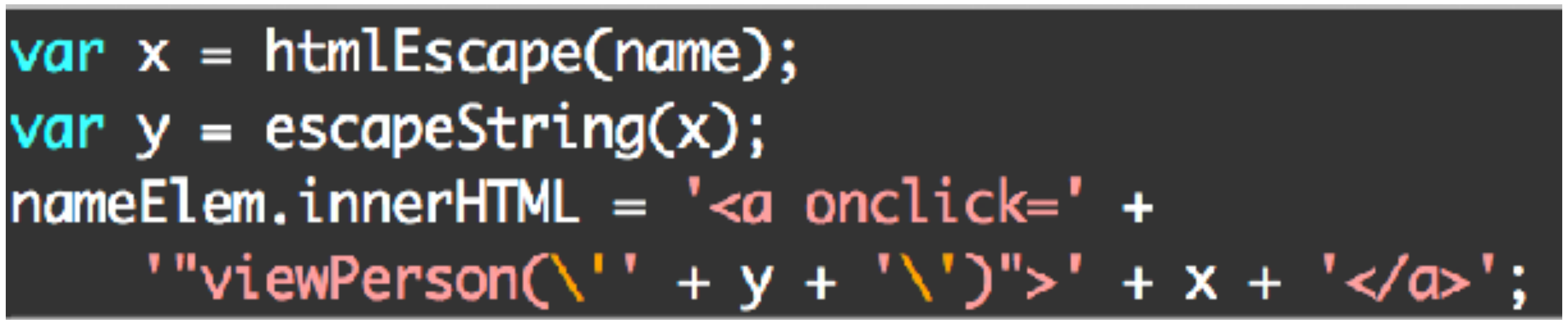
%s/Two/Three/g

Application I: Sanitisers



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

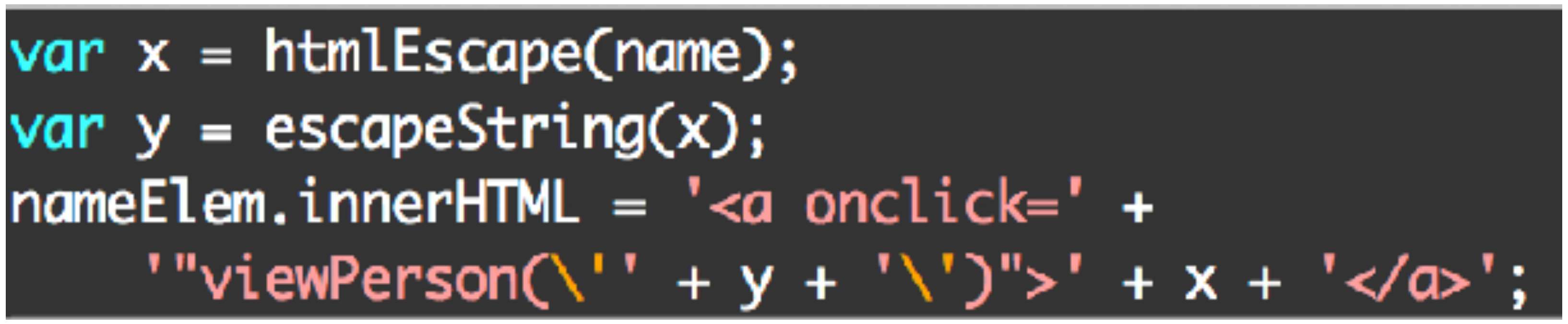
Application I: Sanitisers



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

$$\begin{tikzcd}
\text{<a>} \arrow[r, bend left, "\text{htmlEscape}"] & \text{\&lt;a\&gt;} \arrow[l, bend left, "\text{innerHTML}"]
\end{tikzcd}$$

Application I: Sanitisers



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

$$\begin{tikzcd}
\text{<a>} \arrow[r, bend left, "\text{htmlEscape}"] & \text{\&lt;a\&gt;} \arrow[l, bend left, "\text{innerHTML}"]
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Tom's} \arrow[r, "\text{escapeString}"] & \text{Tom\textbackslash's}
\end{tikzcd}$$

Application II: Web Templating

Application II: Web Templating

<u>HTML template (with Mustache)</u>

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

Application II: Web Templating

<u>HTML template (with Mustache)</u>

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

<u>JSON files</u>

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

Application II: Web Templating

HTML template (with Mustache)

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

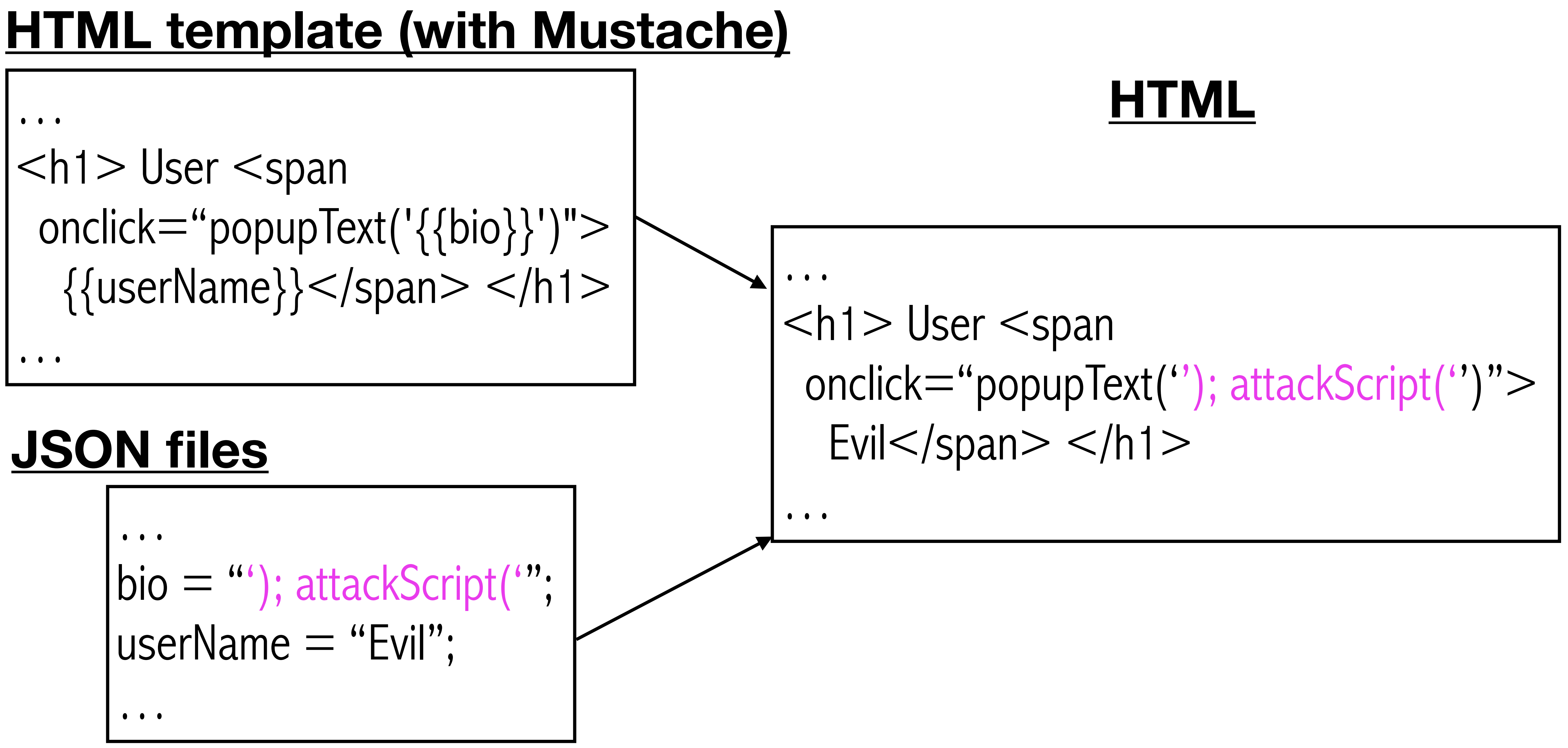
JSON files

```
...
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

$\boxed{\text{x} = \textbf{replaceAll}(\textit{subject},\textit{pat},\textit{rep})}$

replaceAll in String Theory

$$\boxed{x = \textbf{replaceAll}(\mathit{subject},\mathit{pat},\mathit{rep})}$$

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

replaceAll in String Theory

$$\boxed{x = \textbf{replaceAll}(\textit{subject}, \textit{pat}, \textit{rep})}$$

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)

replaceAll in String Theory

$\boxed{x = \textbf{replaceAll}(\textit{subject},\textit{pat},\textit{rep})}$

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,’,\’)

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

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

where

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

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

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

where

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

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

$$\boxed{\begin{array}{l} S ::= \quad y := f(x_1, \ldots, x_n) \mid \textbf{assert}(g(x_1, \ldots, x_n)) \mid S_1; S_2 \\ \qquad\qquad\qquad\qquad \text{where} \\ \quad f : (\Sigma^*)^n \to \Sigma^* \qquad\qquad g : (\Sigma^*)^n \to \{0, 1\} \end{array}}$$

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’)* )
```

Symbolic Execution

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

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

where

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

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’)* )
```

Symbolic Execution

Path Feasibility = Satisfiability (in disguise)

“Straight-Line” Framework

[Lin&Barcelo,POPL’16]

Consider a symbolic execution in a program

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

where

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

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’)* )
```

Symbolic Execution

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

Formula (use SSA form)

<u>Path Feasibility = Satisfiability (in disguise)</u>

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''

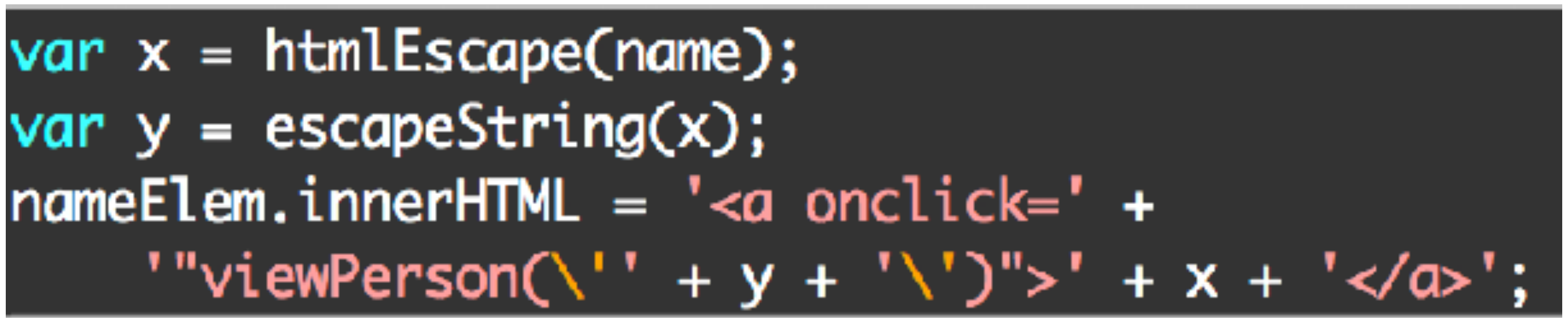
- Captures constraints from symbolic execution in practice

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>';
```

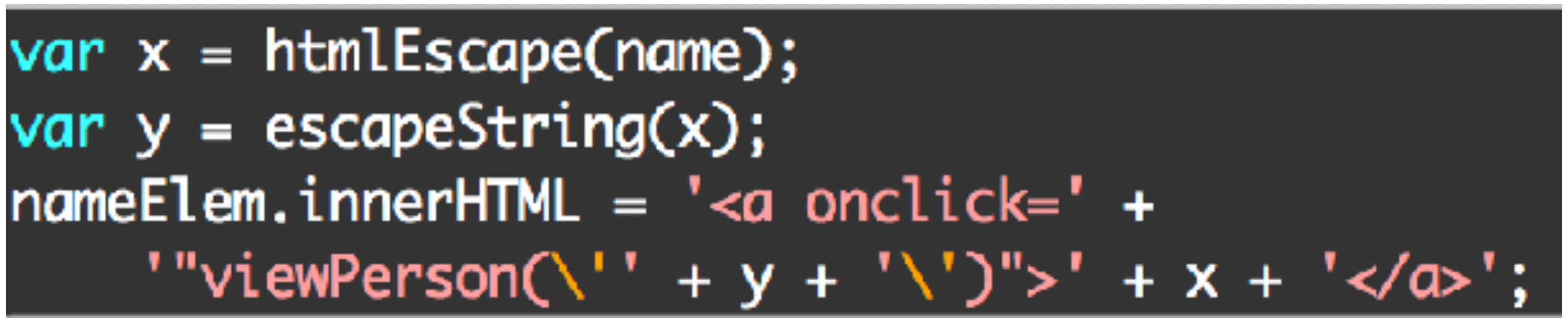
assert(nameElem matches
‘<a onclick=“viewPerson(‘’); attackScript();……”> …… </a>)

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
‘<a onclick=“viewPerson(‘’); attackScript();……”> …… </a>)

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 <u>auto-sanitisation in web templates</u>!

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).

$$X = Y + \text{'aba'} + Y + Z \longrightarrow \begin{array}{l} X0 = \text{replaceAll}(\text{'yabayz'},y,Y) \land \\ X\ = \text{replaceAll}(X',z,Z) \end{array}$$

Proof Idea

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

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

Proof Idea

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

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

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

pat = ‘9’

Proof Idea

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

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

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

pat = ‘9’

$\mathbf{replaceAll}_{\text{pat}}^{-1}(L)$ contains:

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-Za-z]* is a nice name'

pat = '9'

$\mathbf{replaceAll}^{-1}_{\text{pat}}(L)$ contains:

$L \times \Sigma^*$ when 9 doesn't occur in *v*

Proof Idea

Definition: The pre-image of a language *L* under **replaceAll**pat with pattern pat is:

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

L = 'Hi [A-Z][a-z]*, [A-Za-z]* is a nice name'

pat = '9'

$\mathbf{replaceAll}_{\mathrm{pat}}^{-1}(L)$ contains:

$L \times \Sigma^*$ when 9 doesn't occur in *v*

v = '9 is a nice name' *w* matches 'Hi [A-z][a-z]*, [A-Za-z]*'

Proof Idea

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

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

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

pat = ‘9’

$\mathbf{replaceAll}_{\text{pat}}^{-1}(L)$ contains:

$L \times \Sigma^*$ when 9 doesn’t occur in *v*

v = ‘9 is a nice name’ *w* matches ‘Hi [A-z][a-z]*, [A-Za-z]*’

v = ‘Hi 9, 9 is a nice name’ *w* matches ‘[A-z][a-z]*’

Proof Idea

<u>Definition</u>: The <u>pre-image</u> 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-Za-z]* is a nice name’

pat = ‘9’

$\mathbf{replaceAll}^{-1}_{\text{pat}}(L)$ contains:

$L \times \Sigma^*$ when 9 doesn’t occur in *v*

v = ‘9 is a nice name’ *w* matches ‘Hi [A-z][a-z]*, [A-Za-z]*’

v = ‘Hi 9, 9 is a nice name’ *w* matches ‘[A-z][a-z]*’

There are many other combinations!

Proof Idea

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S x S'*.

Proof Idea

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S x S'*.

Why does this imply decidability?

Proof Idea

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S x S'*.

Why does this imply decidability?

```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
z   = replaceAll(x0,'b',y);
assert( z in L2 );
```

Proof Idea

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S x S'*.

Why does this imply decidability?

```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
z   = replaceAll(x0,'b',y);
assert( z in L2 );
```

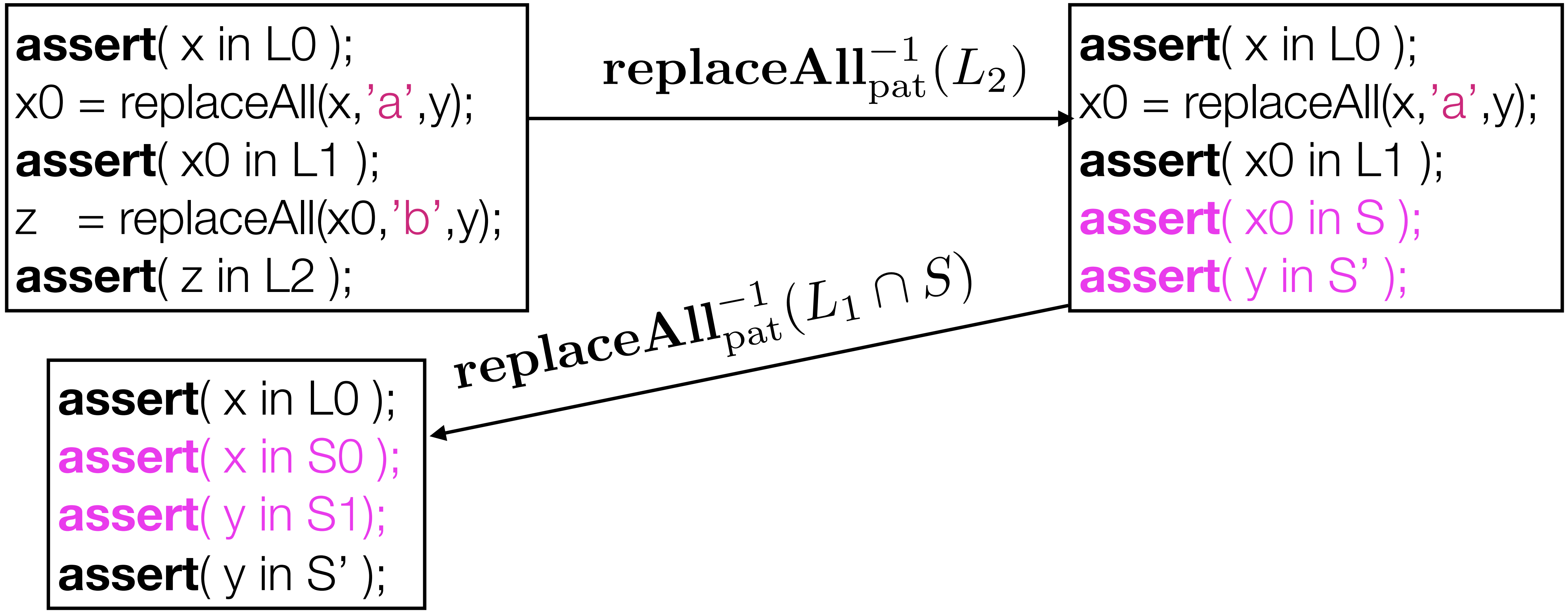
$$\xrightarrow{\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L_2)}$$

```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
assert( x0 in S );
assert( y in S' );
```

Proof Idea

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S x S'*.

Why does this imply decidability?


$\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L_2)$
$\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L_1 \cap S)$

```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
z   = replaceAll(x0,'b',y);
assert( z in L2 );
```

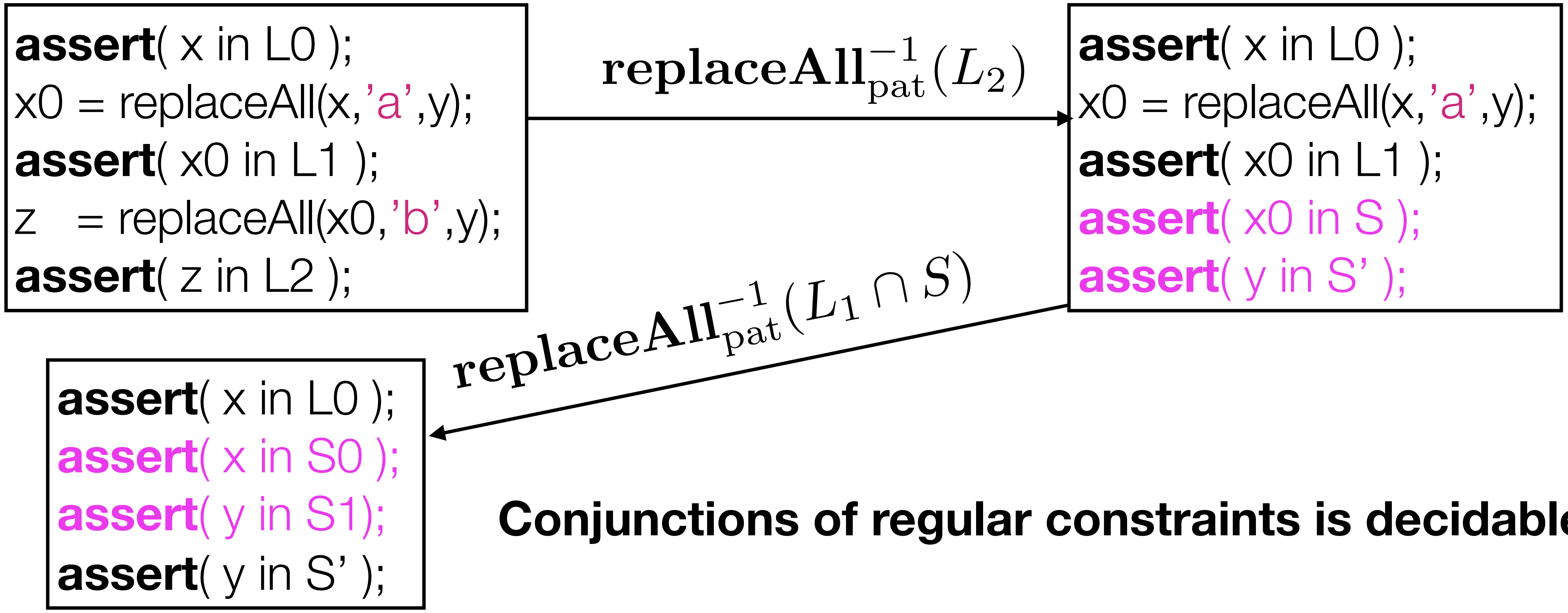
```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
assert( x0 in S );
assert( y in S' );
```

```
assert( x in L0 );
assert( x in S0 );
assert( y in S1);
assert( y in S' );
```

Proof Idea

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S x S'*.

Why does this imply decidability?


$\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L_2)$
$\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L_1 \cap S)$

```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
z  = replaceAll(x0,'b',y);
assert( z in L2 );
```

```
assert( x in L0 );
x0 = replaceAll(x,'a',y);
assert( x0 in L1 );
assert( x0 in S );
assert( y in S' );
```

```
assert( x in L0 );
assert( x in S0 );
assert( y in S1);
assert( y in S' );
```

Conjunctions of regular constraints is decidable!

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S* x *S'*.

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

pat = '9'

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S* x *S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S* x *S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S* x *S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Observation: finitely many resulting NFA S

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}_{\mathrm{pat}}^{-1}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S* x *S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Observation: finitely many resulting NFA S

Step 2: Extract regular constraint S' for replacement string

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}_{\mathrm{pat}}^{-1}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S x S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Observation: finitely many resulting NFA S

Step 2: Extract regular constraint S' for replacement string

Each 9-labeled transition in L contributes to S'

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S* x *S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Observation: finitely many resulting NFA S

Step 2: Extract regular constraint S' for replacement string

Each 9-labeled transition in L contributes to S'

subject = 'Hi 9, 9 is a nice name'

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}_{\mathrm{pat}}^{-1}(L)$ is a <u>recognisable</u> set, i.e., a finite union of products of regular languages *S x S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Observation: finitely many resulting NFA S

Step 2: Extract regular constraint S' for replacement string

Each 9-labeled transition in L contributes to S'

subject = 'Hi 9, 9 is a nice name'

S' = '[A-z][a-z]*' ∩ '[A-za-z]*'

Proof Idea of Lemma

Lemma: $\mathbf{replaceAll}^{-1}_{\mathrm{pat}}(L)$ is a recognisable set, i.e., a finite union of products of regular languages *S x S'*.

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

pat = '9'

Step 1: Guess where 9 could appear in subject (w.r.t. L)

nondeterministically add transitions with label 9 to NFA L

Observation: finitely many resulting NFA S

Step 2: Extract regular constraint S' for replacement string

Each 9-labeled transition in L contributes to S'

subject = 'Hi 9, 9 is a nice name'

S' = '[A-z][a-z]*' ∩ '[A-za-z]*'

= '[A-z][a-z]*'

Complexity Consideration

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

Complexity Consideration

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

Best lower bound is PSPACE-hardness

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

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

z = **replaceAll**(x,'a',x)

prohibits this crazy use

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 <u>length constraints</u>

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

$\boxed{\textbf{assert(len(x) = len(y))}}$

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

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

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

Variable in the Pattern

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

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

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,regex,VAR)
- String solver based on our constraint language

https://github.com/TinyYan/z3-replaceAll