Some Challenges in Algorithmic Verification of String-Manipulating Programs

Anthony W. Lin
(TU Kaiserslautern and MPI, Kaiserslautern)

TPSS'21
Invited Talk

Goal

Describe some of current challenges in string constraint solving that arise from applications in string analysis of programs.

String analysis of programs:

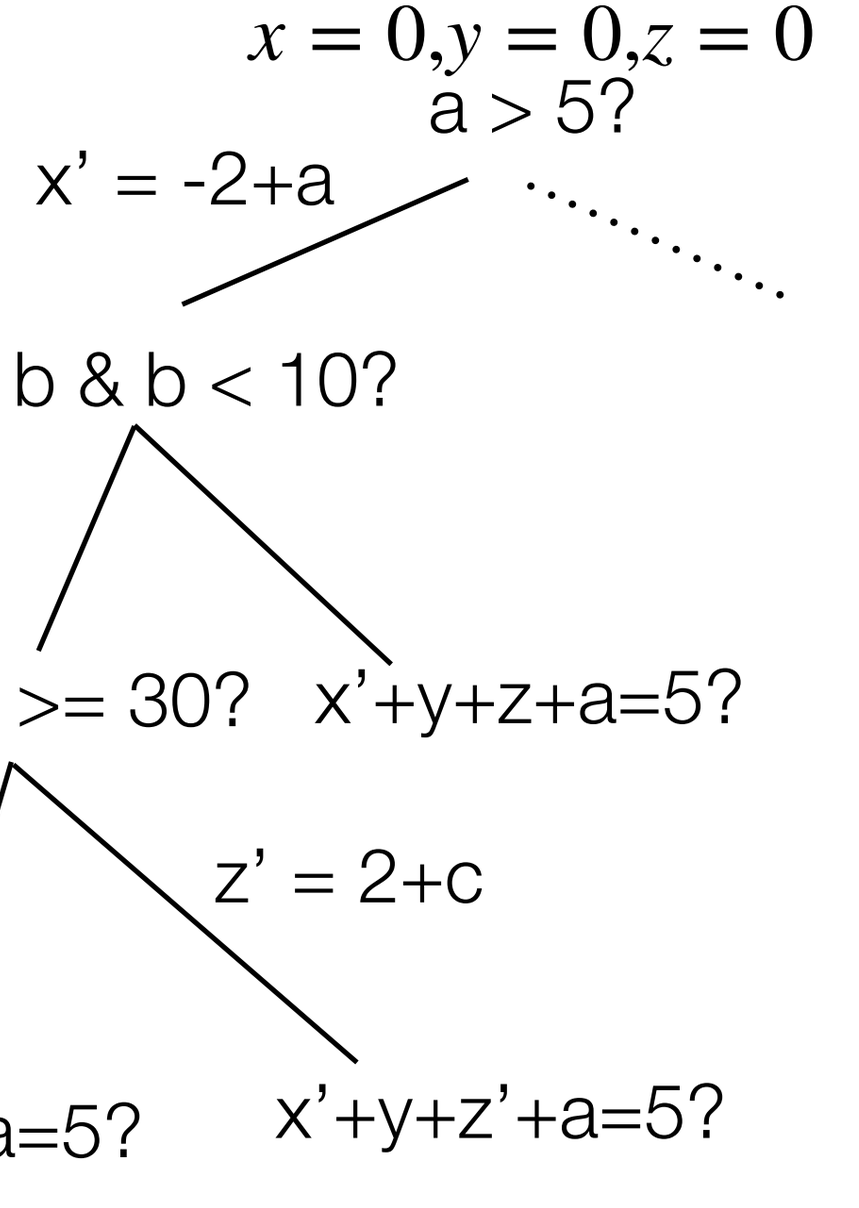
1. Symbolic execution
2. Invariance checking

Symbolic Execution

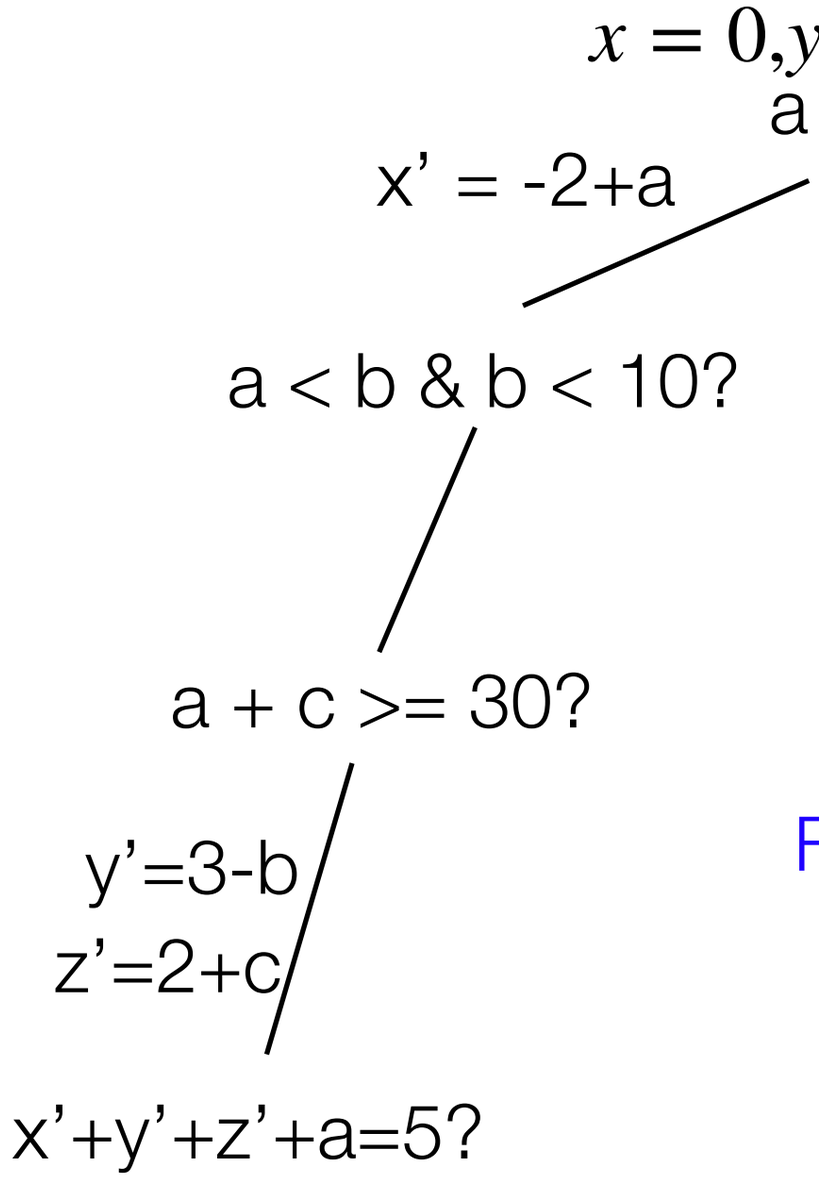
Program

```
1: def foo(int a, int b, int c):
2:     int x, y, z = 0, 0, 0
3:     if a > 5:
4:         x = -2+a
5:     if a < b and b < 10:
6:         if a + c >= 30:
7:             y = 3-b
8:         z = 2+c
9:     assert x+y+z+a != 5
```

Symbolic Execution Tree



Symbolic Execution



Conjunct all constraints along the path

$$x = 0 \wedge y = 0 \wedge z = 0 \wedge a > 5 \wedge x' = -2+a \wedge a < b \wedge b < 10 \wedge a + c >= 30 \wedge y' = 3-b \wedge z' = 2+c \wedge x'+y'+z'+a = 5$$

Formula in quantifier-free theory of linear arithmetic

Decidable (NP-complete)

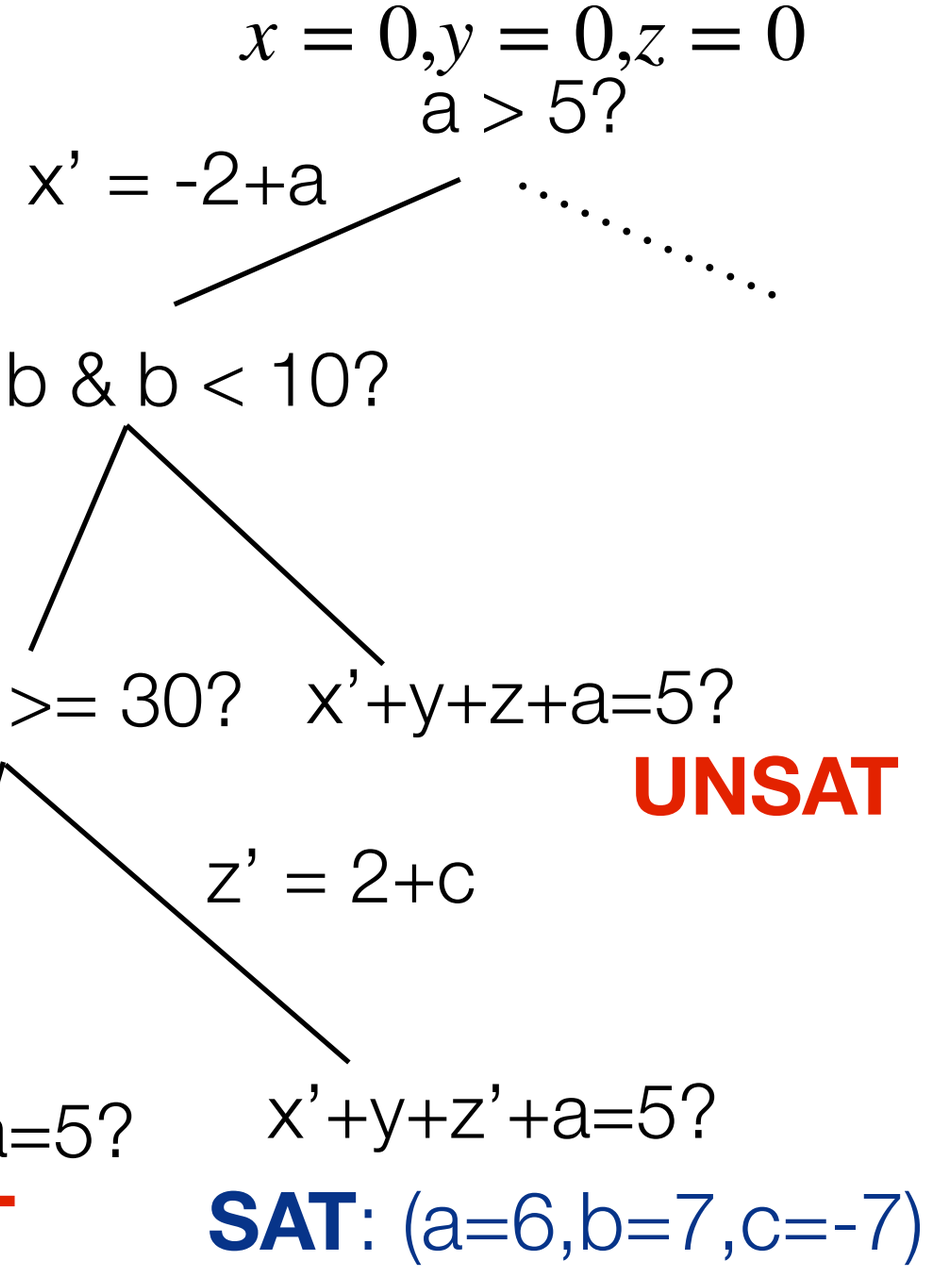
Fast SMT solver

Symbolic Execution

Program

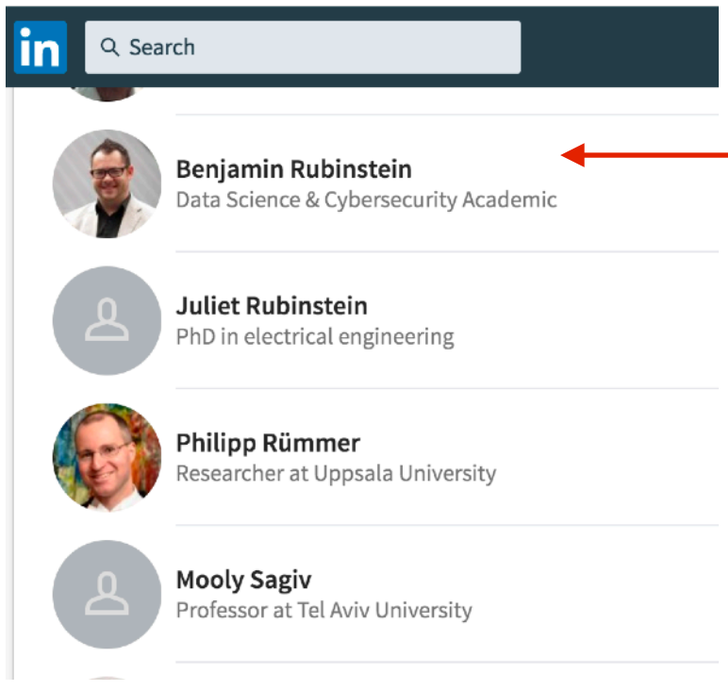
```
1: def foo(int a, int b, int c):
2:     int x, y, z = 0, 0, 0
3:     if a > 5:
4:         x = -2+a
5:     if a < b and b < 10:
6:         if a + c >= 30:
7:             y = 3-b
8:         z = 2+c
9:     assert x+y+z+a != 5
```

Symbolic Execution Tree



Can we do the same with string-manipulating programs?

Example



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?

Motivation

- Strings are a fundamental data type in programming languages, esp. in popular languages like JavaScript, Python, etc.

- Many subtle bugs (some could have serious security consequences, e.g., XSS, code injection) are caused by string manipulation

- Perhaps the most actively investigated theory in SMT (e.g. 30 solvers developed in the past ~10 years alone)

- A long and beautiful history in logic and computation with many as yet unsolved problems

Among many solvers …

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

Which String Theory?

Domain = set of all strings

Quantifier-Free Theories (as common in SMT)

Problem 1: *Which string operations should we allow in the theory?*

S*ome useful* string operations:

1. String concatenation
2. Regex matching (a.k.a. regular constraint).
3. Length constraints
4. Replace (first occurrence, all occurrence, etc.)
5. Transductions (e.g. toUpper, escape, …)
6. Substring/Indexof/CharAt
7. match + regex with capture groups?
8. String-number conversions
9. …

Which String Theory?

Problem 2: *what is a letter?*

An answer: take a finite alphabet like in automata theory

In practice, such a finite alphabet is large (e.g. unicode)

1F600	😀	grinning face
1F601	😁	grinning face with smiling eyes
1F602	😂	face with tears of joy

Although this doesn’t affect decidability, this raises important implementation questions.

SMT over Strings

After many years of disagreement, an SMT file format for string theory was formalized last year: http://smtlib.cs.uiowa.edu/theories-UnicodeStrings.shtml

Some highlights:

1. Concatenation, length, regex matching, string-number conversions, replaceall are added
2. Unicode alphabet

Theorem: SMT over strings (as specified above) is undecidable!

Main Challenge

Come up with decidable fragments of string theories. Delineate the boundary of decidability, and pinpoint computational complexity. Develop/implement good string solving algorithms for these fragments.

A Logic for Symbolic Execution

Symbolic Execution

Symbolic execution is a sequence of assignments/assertions:

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

where

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

A symbolic execution is a formula in disguise:

```
x := x.aba.y;
y := replaceAll(x,a,c);
assert( y in b* )
```

Symbolic Execution

```
x1 = x.aba.y /\
y1 = replaceAll(x1,a,c) /\
y1 in b*
```

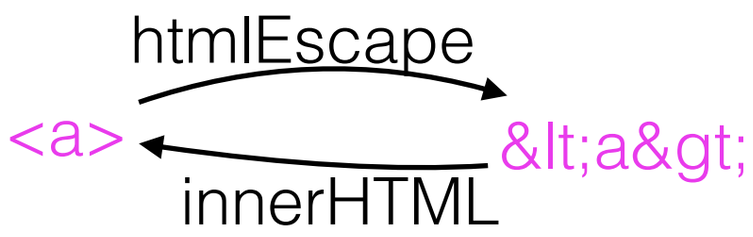
Formula in string theory

Path Feasibility Problem: decide if there exist input strings whose execution satisfies all the assertions

Likewise, this is just satisfiability in disguise

Example

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



Reduce to Path Feasibility

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

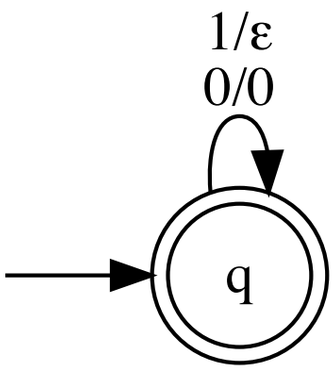
nameElem has to match

e1 = ……

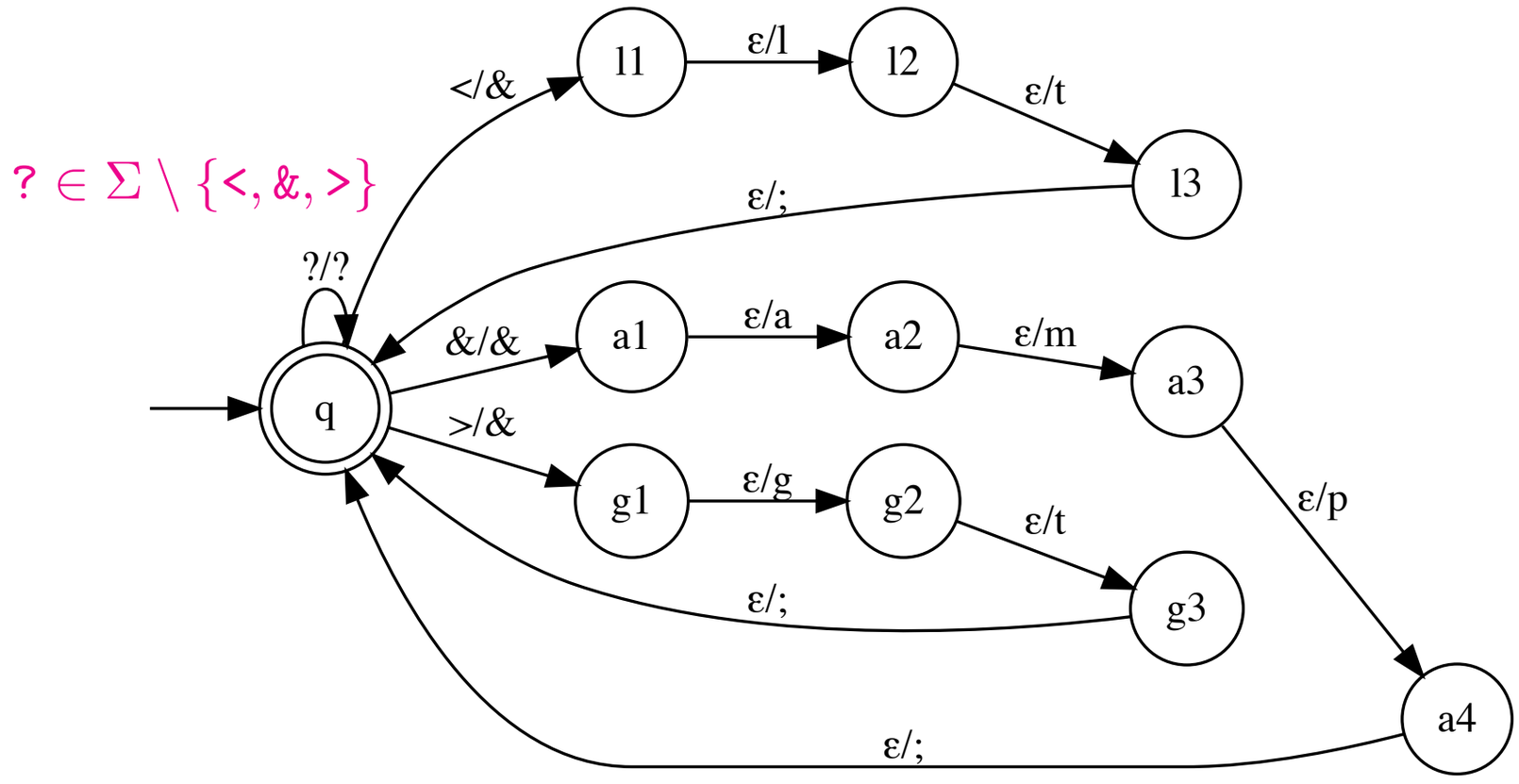
```
x := R1(name);
y := R2(x);
z := w1 . y . w2 . x . w3;
nameElem_innerHTML := R3(z);
assert( nameElem_innerHTML matches e1 )
```

These R1, R2, and R3 can be captured by finite transducers, or finitely many applications of replaceAll

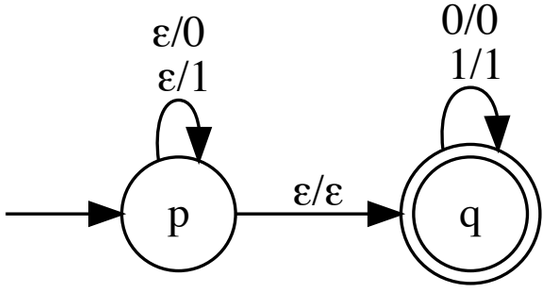
Examples of Transducers



Erase all occurrences of 1



Replace: < by <, > by >, and & by &



Input is a suffix of output

Transducer models for htmlEscape, innerHTML, … exist but more complex

Decidability

Case 1: theory of concatenation (a.k.a. word equations)

Operations: concatenation

Conditionals: string equality, and regular constraints

Decidable [Makanin'77, Schulz'90], and

PSPACE-complete [Plandowski'00,Jez'16]

Decidable with length constraints (e.g. |x| = |y|) is open

This theory is supported by most solvers (though not completely)

This does not capture our example above, and many other examples from web applications (eg, which require encode/decode)

Undecidability

Proposition: Path Feasibility with equality, regex matching, 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''

The Straight-Line Framework

(L. & Barcelo 2016)

Inspired by (1) Solved-form Constraints (Ganesh et al.'12),
(2) Acyclic constraints over rational relations (Barcelo et al.'12)

Developed further by Chen et al., Abdulla et al., etc.

Main Benefit: capture most constraints from real-world programs with many kinds of string functions, while allowing many decidability

Hypothesis (Strong Version)

(Using reformulation of Chen at al.'19)

Assertions are expressible as a bool. combination of regular constraints

$$(x \in aa^* \vee y \notin (bab^* + a^*)) \wedge x \in (a^7)^*$$

Hypothesis (Strong Version)

(Using reformulation of Chen at al.'19)

Assertions are expressible as a bool. combination of regular constraints

√

```
x := x . aba . y;
y := replaceAll(x,a,c);
assert( y in b* )
```

√

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

assert(nameElem matches

‘ ……)

X

```
v := x . ab . y
w := y . ba . x
assert( v == w )
```

X

```
assert( len(v) == len(w) )
```

√

```
assert( len(v) >= 5 )
```

Metatheorem

Decidable path feasibility is possible for a rich class of string functions, under this regularity hypothesis.

This is obviously not true for any string function!!

Here is one extra assumption that ensures decidability:

(BClos) Regular constraints are closed under taking pre-image of ***f***

i.e. L is regular => $f^{-1}(L)$ is effectively a bool. comb of regular constraints

BClos is satisfied by lots of string functions: concatenation, replaceAll, many kinds of transducers, …

Theorem: Under Hypothesis and (BClos), path feasibility is decidable

Concat. satisfies (BClos)

$$. : (\Sigma^* \times \Sigma^*) \to \Sigma^*$$

Proof by *automata splitting*

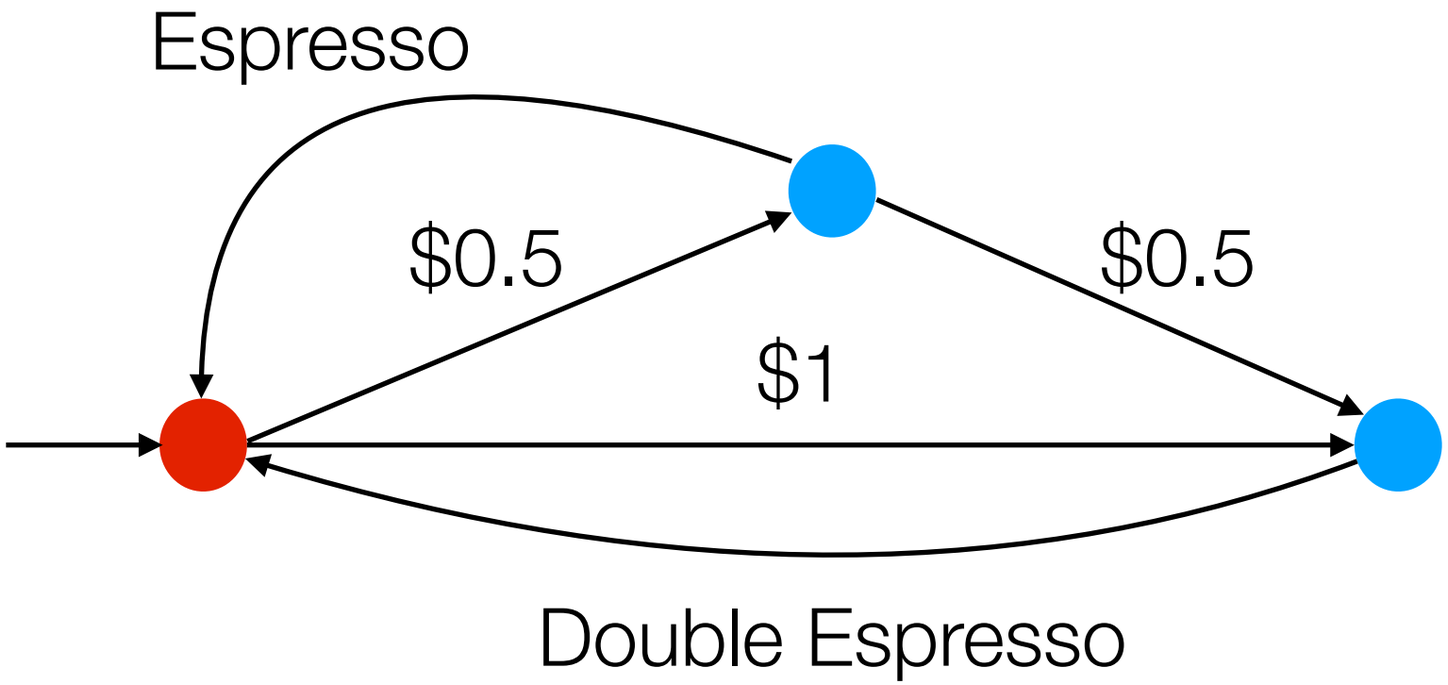
$x \in L \wedge x = y \,.\, z$ and L has states $q_0, \ldots, q_n$

Pre-image of *L* under . is:

$$\bigvee_{i=1}^{n} y \in L_{q_0,q_i} \wedge z \in L_{q_i,q_n}$$

Exercise

$A =$



Consider the constraint $x = yz \wedge x \in L(A)$

Give pre-image of $L(A)$ under the concat above

Decision Procedure (as implemented in OSTRICH)

https://github.com/pruemmer/ostrich

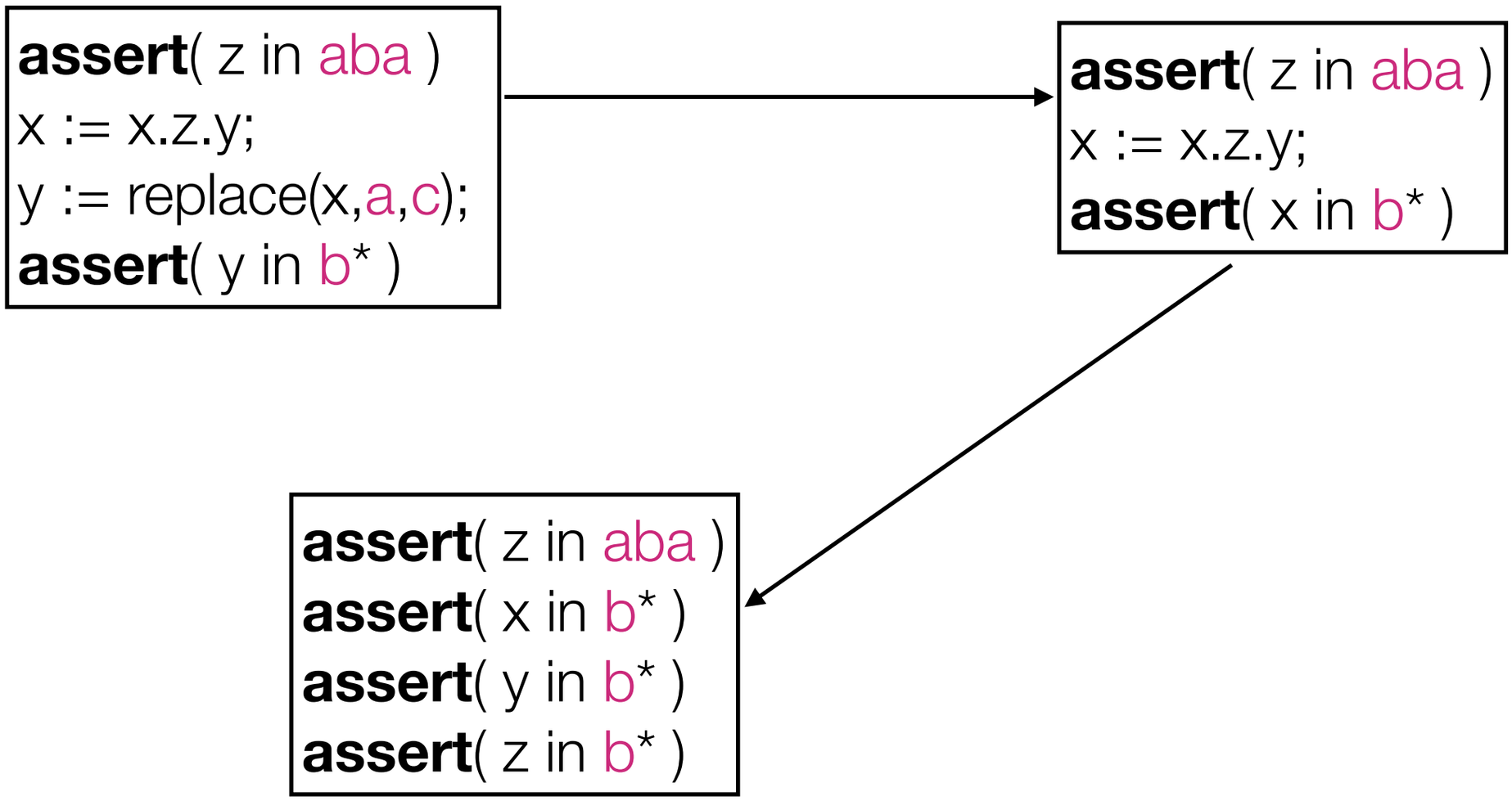
has the following simple recipe …

1. Propagate each individual regular constraint backwards (i.e. in terms of input variables)

2. Solve intersection of regular languages

(See Chen et al'19 for more)

Example



Solve intersection of regular languages (decidable)

Remarks

Weaker version of hypothesis is available, e.g., allowing length constraints (|x| = |y|), or disequality (x != y)

Decidability is still possible for many string functions

(L. &Barcelo'16, Chen et al.'20, Abdulla et. al.'19)

Handling large alphabets (e.g. UTF-16): use symbolic automata/transducers

(see CACM'21 article by D'Antoni and Veanes)

Two Open Problems

Problem 1: Decidability with real-world regular expressions

```
var namesReg = /([A-Za-z]+) ([A-Za-z]+)/g;
var newAuthorList = authorList.replace(nameReg, "$2, $1");
```

Possible unit-test: is #Knuth, Donald#Floyd, Bob# produceable?

Challenges: deterministic matching (greedy/lazy), capture groups, references, … and complex functions exploiting these features (see Loring et al.'19)

Problem 2: Decidability with string-number conversions

In general undecidable (Ganesh & Bezirk'16)

Useful operation (also can be found in SMT-LIB)

MOSCA'19
(Meeting on String Constraints and Applications)

https://mosca19.github.io/

Excellent slides covering other important topics (e.g. by Berzish, Bjorner, Day, Diekert, Jez, Kinder, Majumdar, Murphy, Pasareanu, Tinelli, …)

THANKS

- On Strings in Software Model Checking. In APLAS 2019. (P. Rümmer)
- Graph Logics with Rational Relations and the Generalized Intersection Problem. In LICS’12 (P. Barcelo, D. Figueira, L. Libkin)
- Sound regular expression semantics for dynamic symbolic execution of JavaScript. In PLDI’19 (B. Loring, D. Mitchell, J. Kinder)
- Decision Procedures for Path Feasibility of String-Manipulating Programs with Complex Operations. In POPL'19. (T. Chen, M. Hague, A. Lin, P. Rümmer, Z. Wu)
- String Solving with Word Equations and Transducers: Towards a Logic for Analysing Mutation XSS. In POPL 2016. (A. Lin, P. Barcelo)
- Recompression: A Simple and Powerful Technique for Word Equations. J. ACM’16 (A. Jez)
- Undecidability of a Theory of Strings, Linear Arithmetic over Length, and String-Number Conversion. CoRR 1605.09442/2016 (V. Ganesh and M. Berzish)
- Word Equations with Length Constraints: What’s Decidable? In HVC’12 (V. Ganesh, M. Minnes, A. Solar-Lezama, M. Rinard)
- The Satisfiability of Word Equations: Decidable and Undecidable Theories. In RP’18 (J. Day, V. Ganesh, P. He, F. Manea, D. Nowotka)
- Makanin’s Algorithm. In M. Lothaire: Algebraic Combinatorics on Words. Cambridge University Press, ’01. (V. Diekert)
- Automata Modulo Theories. In CACM’21 (L. D’Antoni, M. Veanes)

ANNEX

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,

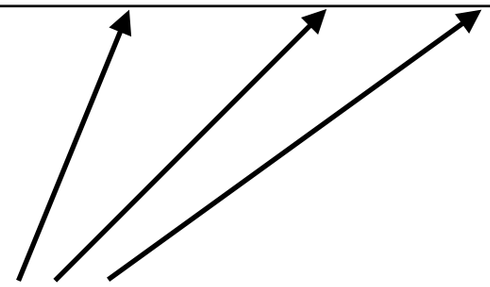
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

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

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)
```

String replacements in HTML templates

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>
...
```

String replacements in HTML templates

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>
...
```