

# The PHP String Analyzer

Yasuhiko Minamide  
University of Tsukuba

# PHP string analyzer

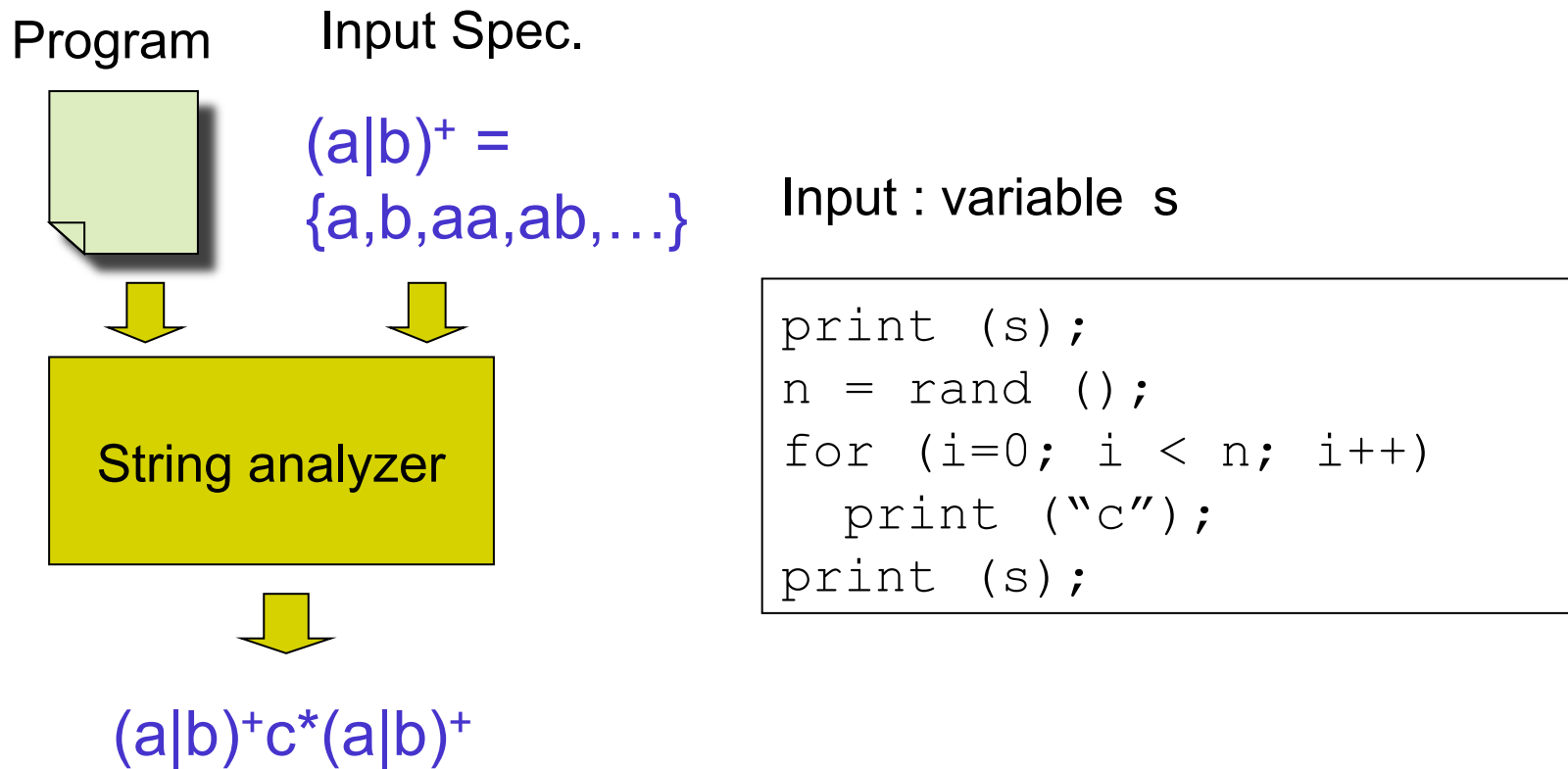
- Approximating string outputs of a program with a context free grammar

 Static approximation of Web pages

- Application
  - Validating dynamically generated (X)HTML documents
  - Detecting vulnerabilities in server-side programs
    - Cross-site scripting, SQL injection
- It is implemented in OCaml

# Java string analyzer

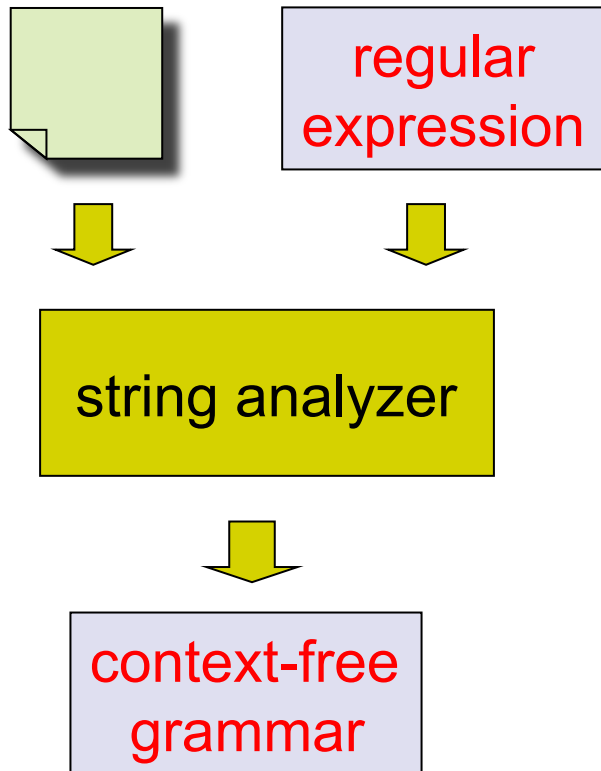
[SAS 2003, Christensen, Møller and Schwartzbach]



Approximation of program's output

# PHP string analyzer

PHP program    input specification



output approximation

- Why PHP?
  - Very popular
  - Many insecure programs
- Why CFG?
  - More expressive
    - HTML validation
  - Natural to implement for CFG
    - Java string analyzer uses the approximation of CFL by RE during analysis

# PHP

```
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<?php
    $i = 0;
    while ( $i < $_POST['number'] ) {
        echo $_POST['string'] . "<BR>\n";
        $i = $i + 1;
    }
?>
</body>
</html>
```

tag for PHP

input from Web browser

string concatenation

# Outline

- What is PHP string analyzer
- How PHP string analyzer works
- Application
  - Validating dynamically generated web pages
  - Detecting cross-site scripting vulnerabilities

# Example

## PHP program

```
<?php
for ($i = 0; $i < $n; $i++)
    $x = '0'.$x.'1';
echo $x;
?>
```

## Input specification

```
$n : int
```

integer

```
$x : /abc|xyz/
```

Reg.exp. "abc" or "xyz"

# Example: result of analysis

```
({ $$30, $$34, $x25 },  
{ 0, 1, a, b, c, x, y, z },  
{ $$30 -> $x25,  
  $$34 -> abc|xyz,  
  $x25 -> $$34|0$x251 },  
$$30)
```

nonterminals

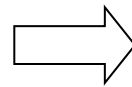
terminals

productions

start symbol

## Simplified version

```
Y -> abc|xyz  
X -> Y|0X1
```



```
{ 0nabc1n } U  
{ 0nxyz1n }
```

# Example with string operations

```
<?php
for ($i = 0; $i < $n; $i++)
    $x = '0'.$x.'1';
echo str_replace("00","0",$x);
?>
```

Grammar obtained by analyzer

$$S \rightarrow abc \mid xyz \mid X1$$
$$X \rightarrow 0abc \mid 0xyz \mid 0S1$$

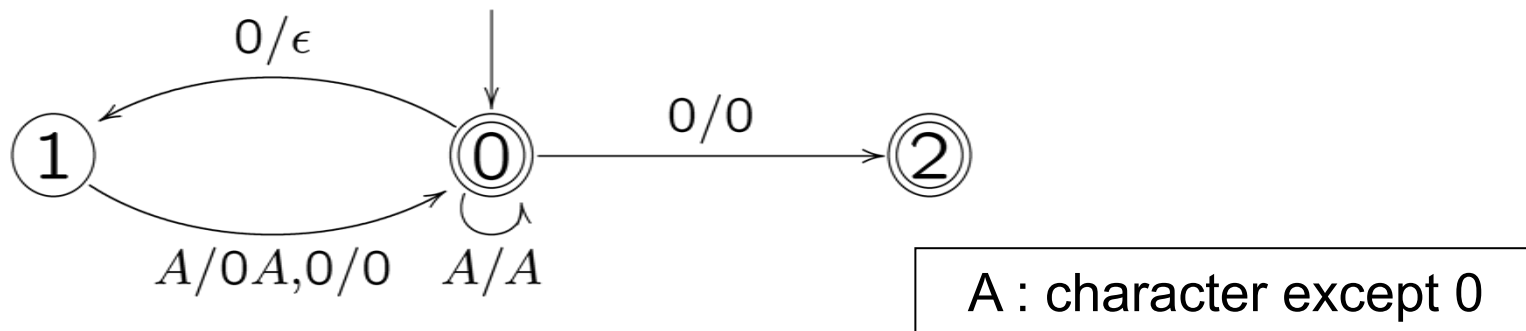
{ abc, 0abc1, 0abc11, 00abc111, ... }  $\cup$

{ xyz, 0xyz1, 0xyz11, 00xyz111, ... }

This precisely approximates output of the program

# Rational transducer

- Automaton with output
- Image of CFL under finite transducer is CFL
- Many string operations can be represented
- Example: `str_replace("00","0",$x)`



- $0/\epsilon$  : output  $\epsilon$  for input 0
- $A/OA$  : output 0A for input A, e.g. output 0a for input a

# Example: image under transducer

`$x`

$X \rightarrow abc \mid xyz \mid 0X1$

`str_replace("00", "0", $x)`



image under the transducer

$S \rightarrow abc \mid xyz \mid X1$

$X \rightarrow 0abc \mid 0xyz \mid 0S1$

- Algorithm : intersection of regular and context-free languages

Note: treatment of string operation in Java string analyzer was ad hoc

# Experiments :

## Grammar obtained by the Analyzer

	lines	CFG		time (sec.)
		nonterminals	productions	
webchess	2224	300	450	0.36
schoolmate	8085	7985	9505	39.92
faqforge	843	180	443	0.16
phpwims	726	82	226	0.13
timeclock	462	656	1233	0.15
tagit	890	858365	6961180	4933.17

Huge grammar for tagit : the size of the grammar can be exponential on the number of string operations.

```
$p = str_replace(' [b] ', '<b>', $p);  
$p = str_replace(' [/b] ', '</b>', $p);  
$p = str_replace(' [i] ', '<i>', $p);  
...
```

# Safety checking

- Checking approximation against output specification
  - Output specification : **regular expression**
- Two modes of checking
  - Checking against safe strings  
 $L(\text{Prog}) \subseteq L(\text{OSpec})$
  - Checking against unsafe strings  
 $L(\text{Prog}) \cap L(\text{OSpec}) = \{\}$

Note: these checks are undecidable if OSpec is written in CFG

# Example:

## Checking against unsafe strings (1/2)

Checking whether the output contains `<script>` tag

```
<html><body>
<?php
$x = $_POST['name'];
//$x = htmlspecialchars($x);
for ($i = 0; $i < 2; $i++)
    echo $x;
?>
</body></html>
```

← sanitization  
(Escaping special characters)

```
htmlspecialchars("<") = "&lt;";"
```

# Example : Checking against unsafe strings (2/2)

Output specification

```
.*<script>.*
```

$L(\text{Prog}) \cap L(\text{OSpec}) = \{\}$ ?

- Without sanitization
  - not disjoint: `<script>` might be in output

Example

```
<html><body>  
<script>  
</body></html>
```

- With sanitization
  - disjoint: `<script>` cannot be included

# Outline

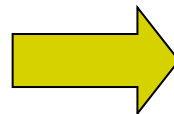
- What is PHP string analyzer
- How PHP string analyzer works
- Application
  - Validating dynamically generated web pages
  - Detecting cross-site scripting vulnerabilities

# Translating output functions

## Translating “echo” into concatenations and assignments

- new variables representing the accumulated output of each program point

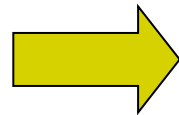
```
$i = 0; $S = "";  
echo "abc";  
while ($i < 10) {  
    $S = $S."xx";  
    $i = $i + 1;  
}  
echo $S;
```



```
i = 0; $S = "";  
$O1 = "abc";  
while ($i < 10) {  
    $S = $S."xx";  
    $i = $i + 1;  
}  
$O = $O1.$S;
```

# Collecting constraints

```
i = 0; $S = "";  
$O1 = "abc";  
while ($i < 10) {  
    $S = $S."xx";  
    $i = $i + 1;  
}  
$O = $O1.$S;
```



```
"" ⊆ S  
"abc" ⊆ O1  
S."xx" ⊆ S  
O1.S ⊆ O
```

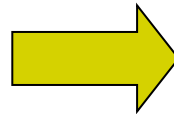
Constraints on  
sets of strings

- Ignoring non-string values

Note: the program is translated into an functional intermediate before collecting constraints

# Translation into CFG

## Constraints

$$\begin{aligned} \text{""} &\subseteq S \\ \text{"abc"} &\subseteq O_1 \\ S.\text{"xx"} &\subseteq S \\ O_1.S &\subseteq O \end{aligned}$$


## CFG

$$\begin{aligned} S &\rightarrow \epsilon \\ O_1 &\rightarrow abc \\ S &\rightarrow Sxx \\ O &\rightarrow O_1S \end{aligned}$$

- Each constraint is translated into a rule
  - This works if the constraints does not contain string operations except concatenation

# Easy case: string operation

- String operations map CFG into CFG
- No string operation in a cycle

Constraints

$" " \subseteq S$   
 $S."xy" \subseteq S$

$\text{str\_replace}("x", "ab", S) \subseteq T$



CFG

$S \rightarrow \varepsilon \mid Sxy$   
 $T \rightarrow \varepsilon \mid Saby$

# Difficult case : string operation

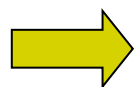
String operation is in a cycle

```
"01" ⊆ S
str_replace("0", "x0y", S) ⊆ T
str_replace("1", "1z", T) ⊆ S
```

Solution

```
S = { xn0yn1zn }
T = { xn+10yn+11zn }
```

Not a CFG



Rough approximation is used

{ 0, 1, x, y, z }<sup>\*</sup>

# Dynamic Features of PHP

- Strings as variable and function names
- Including a file whose names is constructed at runtime

```
function foo ()  
{...}  
function bar ()  
{...}  
$f = "foo";  
if (...) $f = "bar";  
$f ();
```

function name is only  
known at runtime

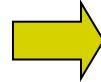
```
$f = "foo";  
if (...) $f = "bar";  
include $f.php;
```

filename is known  
only at runtime

# Analyzing dynamic features

- Iteratively apply string analysis
  - 1 iteration : assume  $\$f = \{\}$  at the application

```
$f = "foo";  
if (...) $f = "bar";  
$f ();
```

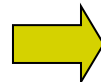


```
$f = "foo";  
if (...) $f = "bar";  
STOP;
```

Result  $\$f = \{\text{"foo"}, \text{"bar"}\}$

- 2 iteration : analyze based on the result above

```
$f = "foo";  
if (...) $f = "bar";  
$f ();
```



```
$f = "foo";  
if (...) $f = "bar";  
foo () | bar ();
```

- terminate analysis if the sets of values that each variable can have do not change

nondeterministic<sup>23</sup>

# Outline

- What is PHP string analyzer
- How PHP string analyzer works
- Application
  - Validating dynamically generated web pages (joint work with Akihiko Tozawa)
  - Detecting cross-site scripting vulnerabilities

# HTML Validation

- HTML Validation
  - Checking validity of a Web page against HTML specification

not valid HTML text: `<p><b>abc</p></b>`

`<ul><p>abc</p></ul>`

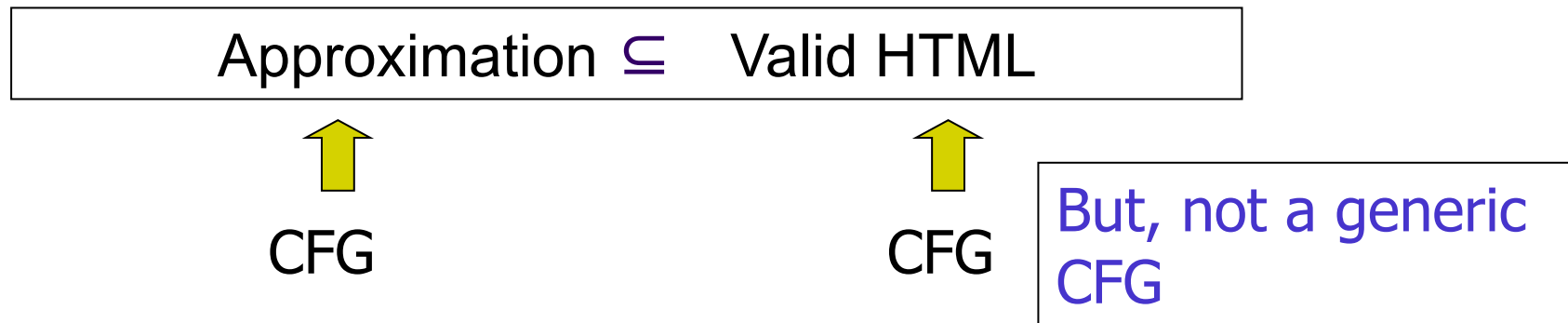
- Our problem: validation of **dynamically generated** Web pages
  - Checking whether a server-side program always generates a valid Web page

**Generated Web pages  $\subseteq$  Valid Web pages**



# Validation (revisited)

Reconsider the validation problem



Valid HTML can be described in a restricted subclass of CFGs

⇒ The inclusion might be decidable

# XML Validation for Context-Free Grammars

[APLAS 06]

Problem: validating a CFG against an XML schema

CFG  $\subseteq$  XML Documents described by a schema

Schema languages: DTD, W3C XML Schema, Relax NG

Our work: two decision algorithms

CFG  $\subseteq$  XML-grammar

CFG  $\subseteq$  regular hedge grammar

# Our decision algorithms

Decision algorithm:  $\text{CFG} \subseteq \text{XML-grammar}$   
(complexity: exponential)

XML-grammars: a subclass of CFGs modeling to DTDs  
[Berstel & Boasson, 2000]

⇒ this enables XHTML validation

Decision algorithm:  $\text{CFG} \subseteq \text{regular hedge grammar}$   
(complexity: doubly exponential)

Regular hedge grammars: tree automata, model of Relax NG

Note: XML-grammars is a subclass of regular hedge grammars

Both algorithms are incorporated into PHP string analyzer

# Experiments : XHTML Validation

Validation of PHP program generating XHTML

program	lines	depth	time	result
WebCalendar	8736	8	3m6s	2 bugs
phpScheduleIt	6960	15	2m19s	valid
marktree	1396	$\infty$	9m40s	2 bugs

## Bugs

- Unmatched start and end tags
- `<a>` occurs under `<ul>`
- `<input ...>` instead of `<input ... />`

Note: we need to modify programs to improve precision of analysis  
we do not check attributes

# Issue on precision of analysis

Analyzer ignores conditional expressions in if-expression

```
if ( $pri == 3 ) echo "<strong>";  
...  
if ( $pri == 3 ) echo "</strong>\n";
```

⇒ rewrite the program so that the start and end tags outputted under the same branch

```
if ( $pri == 3 ) {  
    echo "<strong>";  
    ...;  
    echo "</strong>\n";  
} else  
    ...;
```

```

if ( $login != $event_owner && strlen ( $event_owner ) ) {
  if ($layers) foreach ($layers as $layer) {
    if ( $layer['cal_layeruser'] == $event_owner ) {
      $tmp .= "<span style=¥"color:" . $layer['cal_color'] . ";¥">";
    }
  }
}
.....

```

The conditions for `<span>` and `</span>` do not match

```

if ( $login != $user && $access == 'R' && strlen ( $user ) ) {
  $tmp .= "(" . translate("Private") . ")";
} else if ( $login != $event_owner && $access == 'R' &&
  strlen ( $event_owner ) ) {
  $tmp .= "(" . translate("Private") . ")";
} else if ( $login != $event_owner && strlen ( $event_owner ) ) {
  $tmp .= htmlspecialchars ( $name );
  $tmp .= "</span>"; //end color span
} else {
  $tmp .= htmlspecialchars ( $name );
}

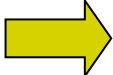
```

# Outline

- What is PHP string analyzer
- How PHP string analyzer works
- Application
  - Validating dynamically generated web pages
  - Detecting cross-site scripting vulnerabilities

# Cross-site scripting vulnerabilities

- Vulnerabilities result from embedding input strings received from a browser into a Web page
  - Input strings might be malicious
  - User may unintentionally send malicious strings
- Prevention: sanitizing input from a browser
  - Converting special characters in HTML into HTML entities

`<script>`  `&lt;script&gt;`

- PHP: htmlspecialchars

# Example : Cross-site scripting

Vulnerable without sanitization

```
<html><body>
<?php
$x = $_POST['name'];
//$x = htmlspecialchars($x);
for ($i = 0; $i < 2; $i++)
    echo $x;
?>
</body></html>
```

← sanitization

Note:

checking just presence of <script> tags does not work because most web pages contains valid <script> tags

# String Analysis + Information Flow Analysis [Wassermann & Su, PLDI 2007]

- Two kinds of strings
  - tainted strings: strings given as inputs
  - untainted strings
- Implementation in PHP string analyzer
  - Information on taint is assigned to **nonterminals** instead of terminal strings

CFG  $S \rightarrow XY, \quad X \rightarrow Y, \quad Y \rightarrow abc$

Language  $\{ abcabc \}$

(red part are tainted)

# Detecting XSS vulnerabilities

Input specification `$_POST` : [**unsafe** /\*/]

```
1: <html><body>
2: <?php
3: $x = $_POST['name'];
4: // $x = htmlspecialchars($x);
5: for ($i = 0; $i < 2; $i++)
6:     echo $x;
7: ?>
8: </body></html>
```

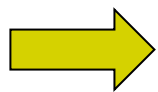
Check: whether the output contains tainted <

# Counter Example: XSS detection

```
<html><body>  
<</body></html>
```

Refinement of counter example generation

- Nonterminals representing an output of strings are annotated with location
- Counter example is generated as a CFG that generates a single string



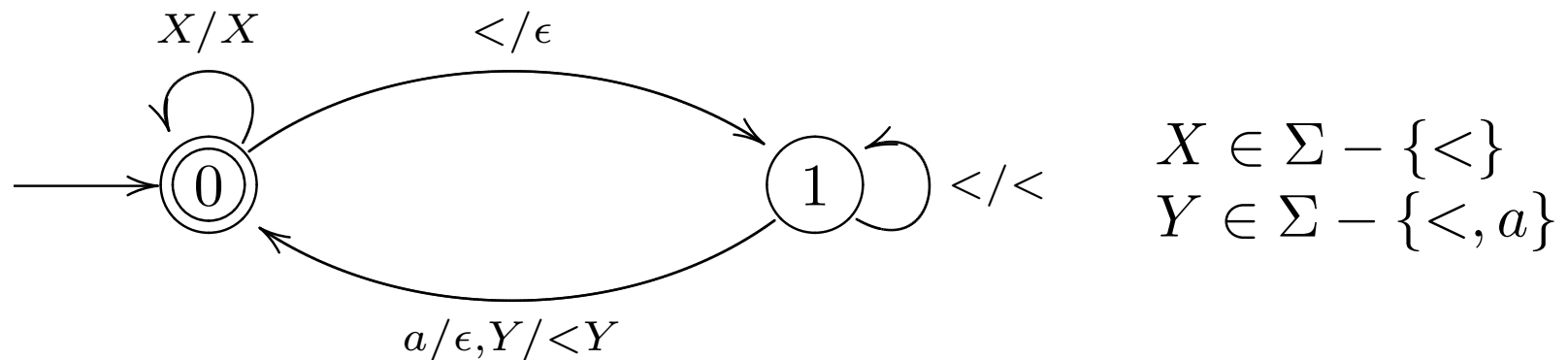
Counter example where we can interactively examine which location each part of the output is generated

# Issue: Vassermann & Su approach

- Information on taint may be lost on string operations

$$S \rightarrow XY, \quad X \rightarrow \langle, \quad Y \rightarrow b$$

replace all occurrences of  $\langle a$  with empty string



Result:

$$S_{0,0} \rightarrow X_{0,1}Y_{0,1}, \quad X_{0,1} \rightarrow \epsilon, \quad Y_{1,0} \rightarrow \langle b$$

Their method is very effective in practice even though this issue

# Future and ongoing works

- Improving precision
  - Conditional expressions
  - Analyzing regular expression matching and replacement precisely
- Detecting command injection attacks
  - theoretically sound method
    - annotating terminals instead of nonterminals
  - precise characterization of web pages causing cross-site scripting