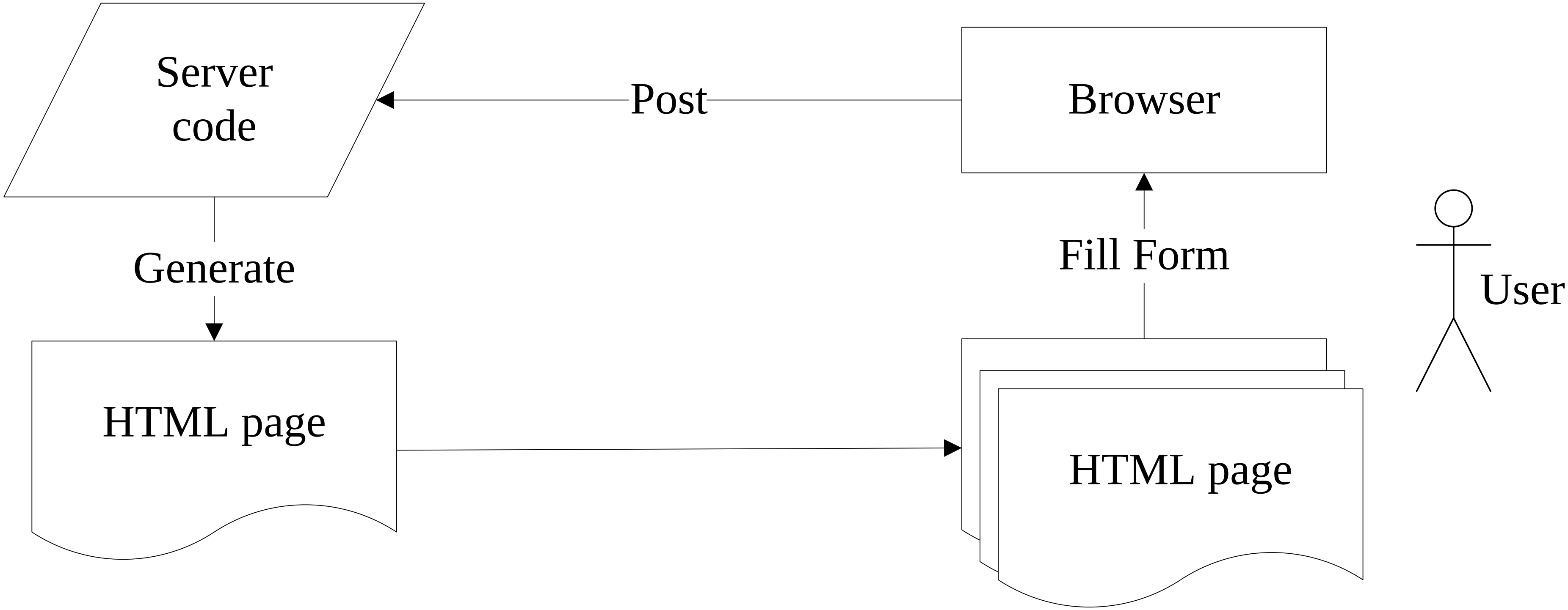
Automating Presentation Changes in Dynamic Web Applications via Collaborative Hybrid Analysis

Xiaoyin Wang*	**UC Berkeley**
Lu Zhang	**Peking University**
Tao Xie	**NC State University**
Yingfei Xiong	**Peking University**
Hong Mei	**Peking University**

*** This work was conducted when Xiaoyin Wang was at Peking University.**

Dynamic Web Application

- Server code generates HTML page according to user inputs



Presentation Changes

- A common task in web application development
  - ✓ Correcting display error or HTML syntax error
  - ✓ Adding interface decorations
  - ✓ Changing appearance styles

- 7% of 600 bug reports investigated are presentation changes

Challenges

- Presentation changes are often identified and reported on **the generated HTML pages**

- Developers have to modify **the server-side code** accordingly

Challenges

Too common for text search

Generated web page:

<p2>~~<tr>~~name:

<input id = 1 color = BFFFFF value = “default”></input></div>country:

<input id = 2 color = BFFFFF value = “country”></input>age:

<input id = 3 color = BFFFFF value = “age”></input><tr>

</p2>

Generation code may scatter

Code generating the web page

$color = BFFFFF; echo “<p2>”; echo “~~<tr>~~”; echo “name:”;

echo “<input id =”.$id.” color = ”.color.” value = “default”></input></div>country:”; $id++;

echo “<input id =”.$id.” color = ”.$color.” value = “country”></input>age:”; $id++;

echo “<input id =”.$id.” color = ”.$color.” value = “age”></input><tr>”; $id++;

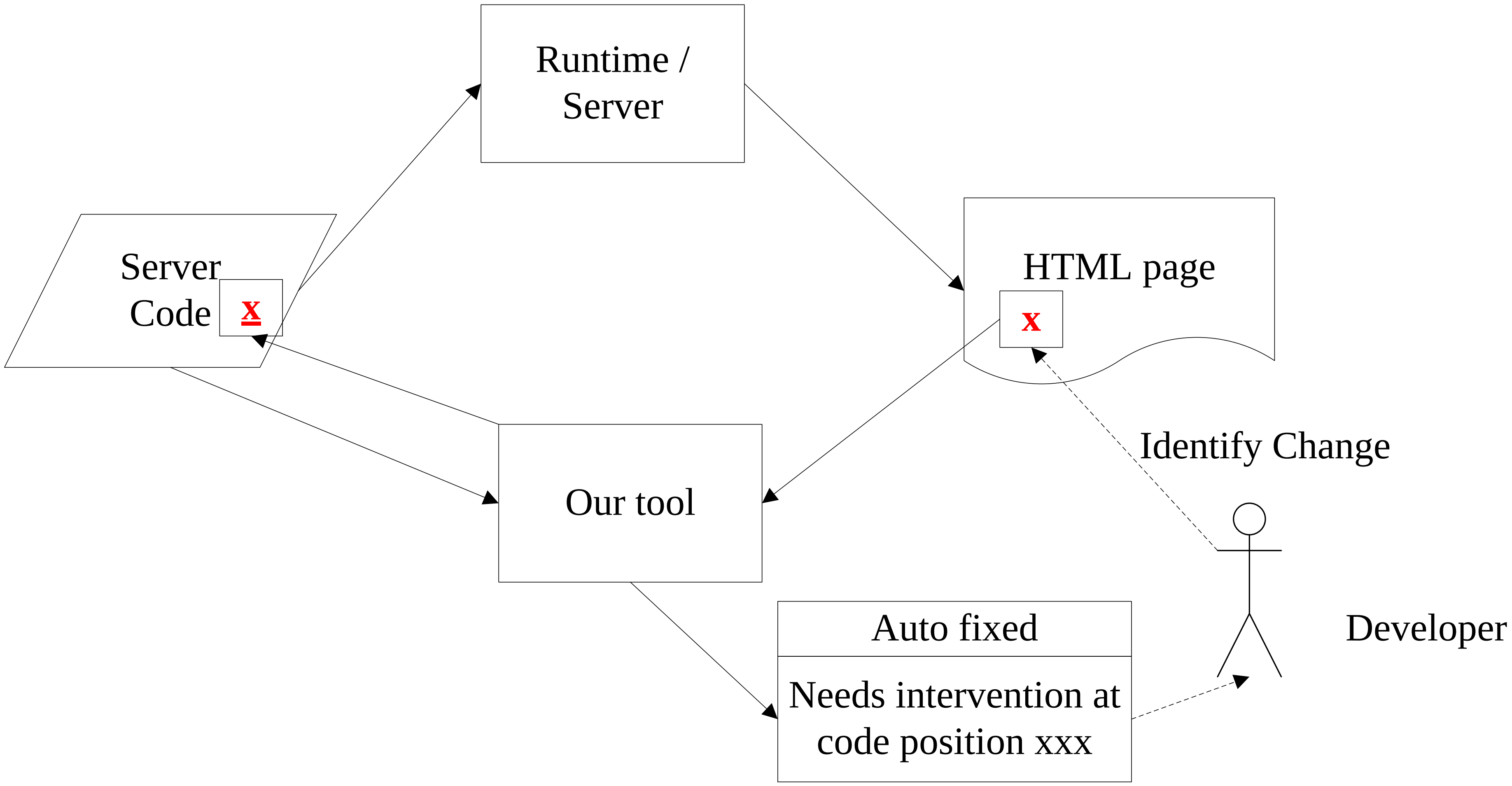
echo “</p2>”;

Affect multiple places

Outline

- Motivation
- Approach
- Empirical Study
- Discussion

Usage Scenario



Runtime / Server

HTML page

Server Code

x

x

Identify Change

Our tool

Developer

Auto fixed

Needs intervention at code position xxx

Approach Overview : Collaborative Hybrid Analysis

- Dynamic String Taint Analysis
  - Locate the piece of code to change
- Static Unexpected Impact Detection
  - Check whether the change is safe

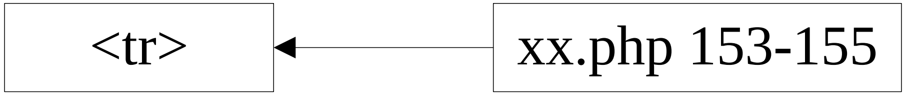
    Safe: perform the change automatically

    Unsafe: report the location to the user

Dynamic String Taint Analysis

- Based on the idea of trace-based bidirectionalization [Xiong et al., ASE07]

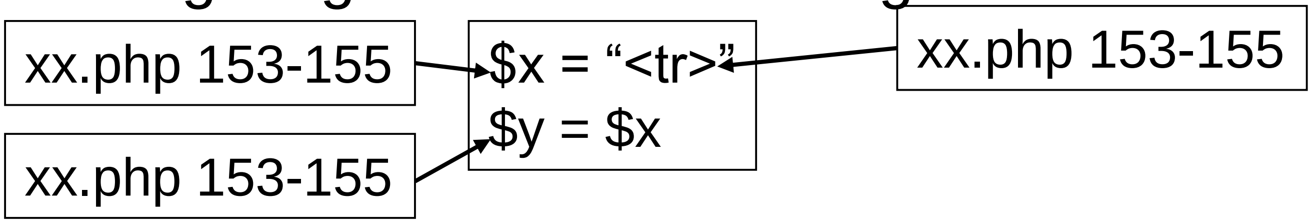
➢ Add a position tag to each constant string and input string



<tr>

xx.php 153-155

➢ Copy the tags together with the strings



xx.php 153-155

$x = “<tr>”

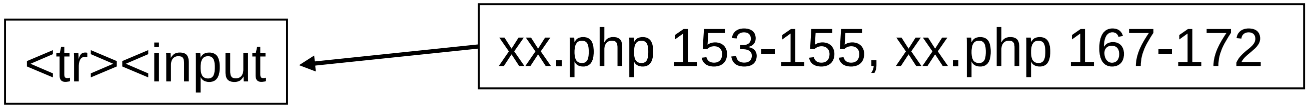
$y = $x

xx.php 153-155

xx.php 153-155

➢ Propagate through string operations

✓ Concatenation



<tr><input

xx.php 153-155, xx.php 167-172

String Operation Handling

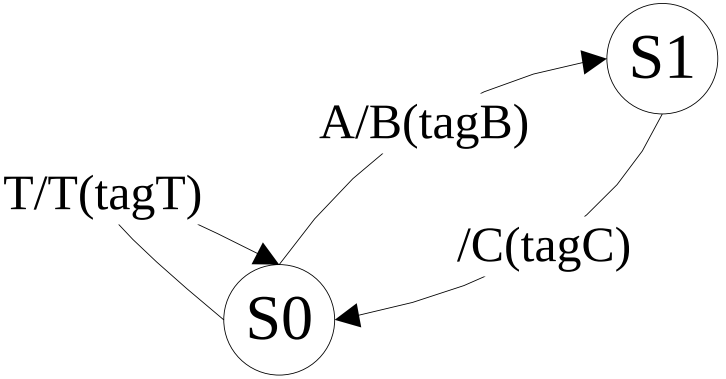
- Problem: do we need to reimplemenet all string operations?
- Solution: working with finite state transducer [Wassermann and Su, PLDI’07]

Constant string A, B, C

String variable $x, $y

$y = B.C

replace($x, A, $y)

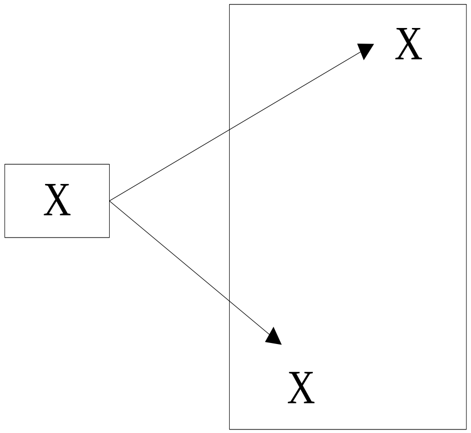


Automatically generated FST with position tag output, based on the runtime value of $y, T = Σ* / AΣ*

Unexpected Impacts

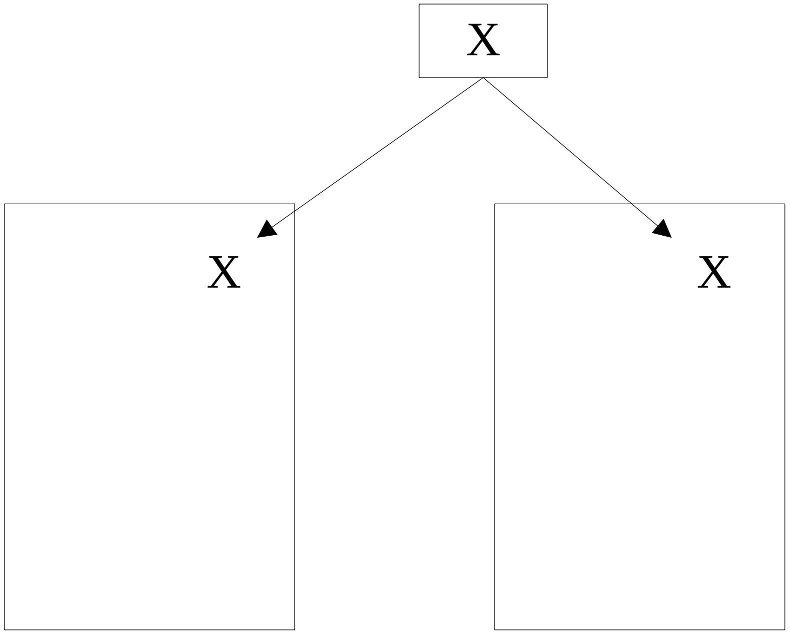
- Inner-page impacts

String origin to be changed affects multiple places in the generated page



- Inter-page impacts

String origin to be changed affects other pages, or contents not generated in this execution



Checking unexpected impacts

- Inner-page impacts

Checking all locations sharing the same string origin are changed consistently

- Inter-page impacts

Checking whether any unexecuted code data-dependent or control dependent on the changed code

Practical Issues

- Insertion:

✓ When a change requires insertion between two variables, human intervention is required

✓ Example:

Code:

```
$title = “contact”;
echo “<td>”.$title. “</td>”
```

HTML:

```
<td>contact␣</td>
```

- Non-constant string origin

✓ When a string origin is not constant (thus cannot be changed directly), human intervention is required

Outline

- Motivation
- Approach
- Empirical Study
- Discussion

Study on the bug reports of three web applications

- 600 Bug Reports from the early history of 3 popular PHP web projects: SquirrelMail, OrangeHRM, and WebCalendar

Project	Start (MM/YY)	End (MM/YY)	KLoc	#Bug Reports	#PC Bug Reports
SquirrelMail	04/00	12/01	8-26	200	7
WebCalendar	06/00	12/02	6-17	200	14
OrangeHRM	03/06	10/06	96-105	200	22

PC Bug Reports: Presentation Change related Bug Reports

Are presentation changes trivial?

- Comparison of processing days between PC Bug Reports and All Bug Reports
- Presentation changes are not trivial (similar processing days compared with other bug reports)

Project / Processing Days	PC Bug Reports		All Bug Reports	
	Avg.	Range	Avg.	Range
SquirrelMail	59.3	0-248	38.8	0- 645
WebCalendar	44.3	0-230	116.5	0-1119
OrangeHRM	20.1	1- 51	18.4	0- 260

Evaluating our approacch

- **Dataset**: 39 presentation change tasks (from 43 reports, in which 4 are duplicate)

- **Evaluation Oracle**: developers' changes

- **Research Questions**:

✓ How effective is our approach on finding the source locations to change?

✓ How effective is our approach on detecting unexpected impacts?

Evaluation Results

Categories	Number of tasks	Percentage
# Correctly Located	39	100.0%
# Automatically fixed	23	59.0%
# Matched fixes	20	51.3%
# Unmatched fixes	3	7.7%
# Human Intervention Required	16	41.0%
# inner-page impact	1	2.6%
# inter-page impact	3	7.7%
# insertions	6	15.4%
# changing non-constants	6	15.4%

Our approach correctly locates all source origins.

Evaluation Results

Categories	Number of tasks	Percentage
# Correctly Located	39	100.0%
# Automatically fixed	23	59.0%
# Matched fixes	20	51.3%
# Unmatched fixes	3	7.7%
# Human Intervention Required	16	41.0%
# inner-page impact	1	2.6%
# inter-page impact	3	7.7%
# insertions	6	15.4%
# changing non-constants	6	15.4%

Most automatic changes match the oracles, yet some do not.

Unmatched Auto-fix

Bug Report No. 1510677 of OrangeHRM

“Feedback information of an operation should be in green when the operation succeeds”

Our approach changed “#FF0000” (red) to “#005500” (green).

Developer change added a check for whether the operation succeeds, and then set different colors

Other unmatched fixes added similar new behavior to the code

Evaluation Results

Categories	Number of tasks	Percentage
# Correctly Located	39	100.0%
# Automatically fixed	23	59.0%
# Matched fixes	20	51.3%
# Unmatched fixes	3	7.7%
# Human Intervention Required	16	41.0%
# inner-page impact	1	2.6%
# inter-page impact	3	7.7%
# insertions	6	15.4%
# changing non-constants	6	15.4%

For the rest of the tasks, our approach correctly identifies the need of human intervention.

Outline

- Motivation
- Approach
- Empirical Study
- Discussion

Limitations

- More suitable for small atomic changes than pervasive or large structure changes

- Currently cannot handle web interface generated with Ajax techniques

- May generate undesirable code changes

Conclusion

- Presentation change being common and non-trivial
- Hybrid approach to presentation changes
  - Dynamic analysis to locate the source code to change
  - Static analysis to ensure the change is safe
- Lightweight approach yet effective

Thanks! Q & A

Evaluation Results

- On locating source code and automatic fixing

Project	#PC tasks	#Locating	#matched auto-fix	#unmatched auto-fix
SquirrelMail	6	6	2	0
WebCalendar	12	12	7	2
OrangeHRM	21	21	11	1
Total	39	39	20	3

Evaluation Results

- On detecting unexpected impacts and practical issues

Project	#PC tasks	#inner-page Impact	#inter-page impact	#insert	#non-constant
SquirrelMail	6	0	0	2	2
WebCalendar	12	1	1	1	0
OrangeHRM	21	0	2	3	4
Total	39	1	3	6	6

Example Task

SquirrelMail ---- Bug #601006: “Rejected e-mail link missing a quote”

Error HTML page:

```
<BR><STRIKE><A HREF="mailto:mymail@gmail.com?
subject=WebCalendar:mycal\>Xiao</a></STRIKE>Rejected";
```

Buggy Code:

```
echo "<BR><STRIKE><A HREF=\"mailto:" . $tempemail ."?
    subject=$subject\>" . $tempfullname . "</a></STRIKE> (" .
    translate("Rejected") . ")\ n";
```

Result of our tool

1. Locate the “\>” in the code as the data origin of the erroneous place in the error HTML page
2. Determine that there is no unexpected impacts and practical issues, so that the fix can be done automatically

Example Task

SquirrelMail ---- Bug #601006: “Rejected e-mail link missing a quote”

Error HTML page:

```
<BR><STRIKE><A HREF="mailto:mymail@gmail.com?
subject=WebCalendar:mycal\>Xiao</a></STRIKE>Rejected";
```

Buggy Code:

```
echo "<BR><STRIKE><A HREF=\"mailto:" . $tempemail ."?
    subject=$subject\>" . $tempfullname . "</a></STRIKE> (" .
    translate("Rejected") . ")\ n";
```

Result of our tool

1. Locate the “\>” in the code as the data origin of the erroneous place in the error HTML page
2. Determine that there is no unexpected impacts and practical issues, so that the fix can be done automatically

Future Directions

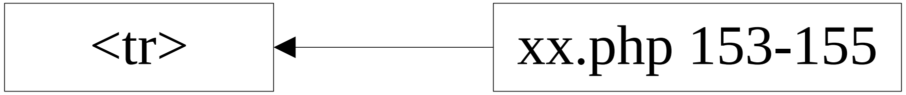
- Empirical studies on more web-based projects
- Handling of more complex presentation techniques, e.g., Ajax
- User study on how much the approach it going to help in real maintenance tasks

Dynamic String Taint Analysis

- Based on the idea of trace-based bidirectionalization [Xiong et al., ASE07]

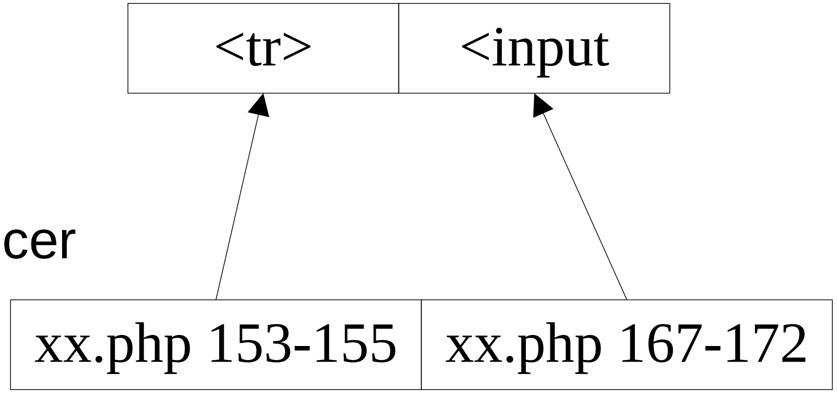
➢ Instrumentation

Add a position tag to each constant string and input string



➢ Propagate through string operations

✓ Concatenation



✓ Other Operation

Simulated with Finite State Transducer

[Wassermann and Su, PLDI'07]

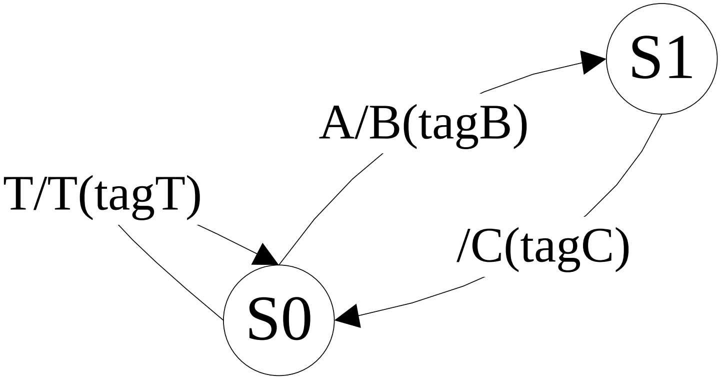
String Operation Handling

Constant string A, B, C

String variable $x, $y

$y = B.C

replace($x, A, $y)



Automatically generated FST with position tag output, based on the runtime value of $y, T = Σ* / AΣ*

Consider A = ‘ts’, $x = ‘abct’(tag1) ‘sdd’(tag2)

Output = ‘abc’(tag1’)B(tagB)C(tagC) ‘dd’(tag2)