Analyzing Configurable Systems with Dynamic Taint Analysis

Miguel Velez, Christian Kästner, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel
Find How Options Influence Control-Flow Decisions
Analyze How Options Interact

Program understanding

Configuration testing

Performance analysis
Exploiting Structure and Behavior of Highly Configurable Systems to Measure Performance

Miguel Velez
Software Engineering Ph.D. student
Carnegie Mellon University
Joint work with: Christian Kästner, Pooyan Jamshidi, and Norbert Siegmund

ConfigCrusher: White-Box Performance Analysis for Configurable Systems

Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, Christian Kästner
Analyze How Options Interact

```
void main(x)
    if(A)
        ...
    if(B)
        foo()
    if(x)
        ...
...

void foo()
    if(C)
        ...
```
Analyze How Options Interact

void main(x)

{A} -> if(A)

{B} -> if(B)
    foo()
    if(x)
    ...

{B,C} -> if(C)

void foo()
Static Taint Analysis
Taint Analysis

```c
void main(x)
{
    if(A)
        ...

    if(B)
    {
        foo()
    }

    if(x)
        ...

    void foo()
    {
        ...
    }

    if(C)
        ...
}
```
Static Taint Analysis

```
void main(x)
  if(A)
    ...
  ...
  if(B)
    foo()
      if(x)
        ...
        ...
  
void foo()
  if(C)
    ...
```

{A} \rightarrow if(A)
{B} \rightarrow if(B)
{B,C} \rightarrow if(C)
Find Configurations to Efficiently Explore Decisions

```c
void main(x)
{A}  if(A)  ...  if(x)  ...
{B}  if(B)  foo()  if(x)  ...
{B,C}  if(C)  ...

void foo()
```

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Find Configurations to Efficiently Explore Decisions

```
void main(x)
if(A)
...

if(B)
foo()
if(x)
...

void foo()
if(C)
...
```

{A} → if(A) → {A}

{B} → if(B) → if(x) → {B, C}

{B, C} → if(C) →

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Overapproximation
SPLat

SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems

Chang Hwan Peter Kim  
University of Texas  
Austin, TX, USA

Darko Marinov$^{1,2}$  
$^1$University of Illinois  
and $^2$Groupon, Inc., USA

Sarfraz Khurshid  
University of Texas  
Austin, TX, USA

Don Batory  

Sabrina Souto  
Paulo Barros  
Marcelo d’Amorim  
Federal University of Pernambuco  
Recife, PE, Brazil

ABSTRACT

Many programs can be configured through dynamic and/or static selection of configuration variables. A software product line (SPL), for example, specifies a family of programs where each program is defined by a unique combination of features. Systematically testing SPL programs is expensive as it can require running each test against a combinatorial number of configurations. Fortunately, a test is often independent of many configuration variables and need not be run against every combination. Configurations that are not example, specifies a family of programs where each program is defined by a unique combination of features (increments in program functionality). Many codebases that power modern websites are also highly configurable. For instance, the code behind the groupon.com website has over 170 boolean configuration variables and can, in theory, be deployed in over $2^{170}$ different configurations.

Systematically testing configurable systems and SPLs is challenging because running each test can, in principle, require many actual executions—one execution for each possibility.
SPLat Infers Control-Flow Interactions

```c
void main(x)
    if(A)
        ...
    if(B)
        foo()
    if(x)
        ...

void foo()
    if(C)
        ...
```

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
SPLat is a Pseudo-Brute-Force Approach

```c
void main(x)
   if(A)
       ...
   if(B)
       foo()
   if(x)
       ...

void foo()
   if(C)
       ...
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Overapproximation
Dynamic Taint Analysis
Iterative Dynamic Taint Analysis
Iterative Dynamic Taint Analysis

Pick initial configuration to execute
Iterative Dynamic Taint Analysis

1. Pick initial configuration to execute
2. Run dynamic taint analysis
Iterative Dynamic Taint Analysis

1. Pick initial configuration to execute
2. Run dynamic taint analysis
3. Derive configurations to explore
Iterative Dynamic Taint Analysis

1. Pick initial configuration to execute
2. Run dynamic taint analysis
3. Derive configurations to explore
4. If not all configurations explored
Iterative Dynamic Taint Analysis

1. Pick initial configuration to execute
2. Run dynamic taint analysis
3. Derive configurations to explore
4. If not all configurations explored, pick new configuration
Iterative Dynamic Taint Analysis

Pick initial configuration to execute

Run dynamic taint analysis

Derive configurations to explore

If not all configurations explored

Pick new configuration
Iterative Dynamic Taint Analysis

1. Pick initial configuration to execute
2. Run dynamic taint analysis
3. Derive configurations to explore
4. If not all configurations explored:
   - Done
   - Pick new configuration
void main(x)
  if(A)
    ...
  if(B)
    foo()
  if(x)
    ...
  ...

void foo()
  if(C)
    ...

Pick Configuration to Execute

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
void main(x)
    if(A) // A=0
        ...
    if(B) // B=0
        foo()
    if(x)
        ...

void foo()
    if(C) // C=0
        ...

 Dynamic Taint Analysis

A B C
0 0 0
Dynamic Taint Analysis

\[
\begin{array}{ccc}
A & B & C \\
0 & 0 & 0 \\
\end{array}
\]

```c
void main(x)
{A}
  if(A) // A=0
    ...
    if(B) // B=0
      foo()
      if(x)
        ...
  ...

void foo()
  if(C) // C=0
    ...
```
Dynamic Taint Analysis

```
void main(x)
{A}  if(A)  //  A=θ
    ...
{B}  if(B)  //  B=θ
    foo()
    if(x)
    ...

void foo()
    if(C)  //  C=θ
    ...
```
Derive Configurations to Explore

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

```c
void main(x)
{
    if(A)
    {
        ...
    }
    if(B)
    {
        foo()
        if(x)
        {
            ...
        }
    }
    void foo()
    {
        if(C)
        {
            ...
        }
    }
}
```

```plaintext
\{A\} \xrightarrow{\text{if}(A)} \text{...} \xrightarrow{\text{if}(B)} \text{foo()}
\xrightarrow{\text{if}(x)} \text{...}
\xrightarrow{\text{void} \text{ foo()}} \text{if}(C) \xrightarrow{\text{...}}
```
Derive Configurations to Explore

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

```c
void main(x)
{
    if(A)
        ...
    if(B)
        foo()
        if(x)
            ...
}

void foo()
{
    if(C)
        ...
}
```
Pick Next Configuration to Execute

\[
\begin{array}{ccc}
A & B & C \\
1 & 1 & 0
\end{array}
\]

\[
\text{void } \text{main}(x)
\]

\[
\text{if}(A) \quad \ldots
\]

\[
\text{if}(B)
\]

\[
\text{foo}() \\
\text{if}(x) \quad \ldots
\]

\[
\text{void } \text{foo}() \\
\text{if}(C) \quad \ldots
\]
# Dynamic Taint Analysis

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```c
void main(x)
    if(A) // A=1
        ...
    if(B) // B=1
        foo()
    if(x)
        ...
    ...

void foo()
    if(C) // C=0
        ...
```
### Dynamic Taint Analysis

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```c
void main(x)
    if(A) // A=1
    ...

if(B) // B=1
    foo()
    if(x)
    ...

void foo()
    if(C) // C=0
    ...
```
Derive Configurations to Explore

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

```
void main(x)

if(A)
...

if(B)
  foo()
  if(x)
    ...

void foo()

if(C)
...
```

{A} → if(A) → A

{B} → if(B) → B
  foo()
  if(x) → B
  0 → 1

{B},{C} → if(C) → B
  C
  1 → 0
  1 → 1

```
void main(x)

if (A)
...

if (B)
foo()
if (x)
...

void foo()

if (C)
...

A
0
1

B
0
1

C
1
0
1
1

Pick Next Configuration to Execute
Dynamic Taint Analysis

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```c
void main(x)
    if(A) // A=0
        ...
    if(B) // B=1
        foo()
        if(x)
            ...

void foo()
    if(C) // C=1
        ...
```
Dynamic Taint Analysis

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```c
void main(x)
{
    if(A) // A=0
        ...
    ...
    if(B) // B=1
        foo()
        if(x)
            ...
    ...
    void foo()
    {
        if(C) // C=1
            ...
    }
```
void main(x)

if(A)

... ...

if(B)

foo()

if(x)

... ...

void foo()

if(C)

... ...

Pick Next Configuration to Execute

{A} → if(A) → A
{B} → if(B) → B
{B},{C} → if(C) → B,C

0 1
0 1
0 1
1 0
1 1
Find Configurations to Efficiently Explore Decisions

```c
void main(x)
{
    if(A)
    ...
    if(B)
    {
        foo();
    }
    if(x)
    ...
}

void foo()
{
    if(C)
    ...
}
```

```
{A}  -->  if(A)  -->  A
    {B}  -->  if(B)  -->  B
          -->  foo()  -->  0  1
          {B},{C}  -->  if(C)  -->  B  C
```

```

A: 0 1
B: 0 1
C: 1 0 1
```
Limitations of Dynamic Taint Analysis

Hard to track implicit flows with unsound analysis
void main()
    x = 0
    if(A)
        x = 1
    if(x)
        ...

Tracking Implicit Flows
Tracking Implicit Flows

```
void main()
    x = 0
    if(A) // A=0
        x = 1
    if(x)
        ...
```
void main()
  x = 0
  if(A)  // A=0
    x = 1
  if(x)
    ...

A

0
void main()
    x = 0
    if(A) // A=0
        x = 1
    if(x)
        ...

Missed information
Limitations of Dynamic Taint Analysis

Hard to track implicit flows with unsound analysis

Tooling overhead
Summary

Dynamic taint analysis

Find how options influence control-flow decisions

Find configurations to efficiently explore decisions