Automated Debugging for Arbitrarily Long Executions

Cristian Zamfir, Baris Kasikci, Johannes Kinder, Edouard Bugnion, George Candea
Debugging is Hard

- Debugging = diagnose + fix the root cause
- May take days-months to diagnose bugs in the real world

1 Concurrency at Microsoft – An Exploratory Survey, CAV workshop 2008
Real World Debugging

Debugging during development
Real World Debugging

Debugging during development

$ gdb ./program
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step
Real World Debugging

Debugging during development  Debugging in the real world

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step
Real World Debugging

Debugging during development

Debugging in the real world

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step

$ ./program
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step

Debugging in the real world

$ ./program
Segmentation fault
Real World Debugging

Debugging during development

```
$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step
```

Debugging in the real world

```
$ ./program
Segmentation fault
$core dumped

$ gdb ./program core
```
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step

Debugging in the real world

$ ./program
Segmentation fault
(core dumped)

$ gdb ./program core
(gdb) reverse-step
Real World Debugging

Debugging during development

$ gdb ./program
(gdb) record
(gdb) run
Segmentation fault
(gdb) reverse-step

Debugging in the real world

$ ./program
Segmentation fault
(core dumped)

$ gdb ./program core
(gdb) reverse-step
Target core command unsupported
Debug Without Recording
Debug Without Recording

What are the classes of information necessary for debugging?
Debug Without Recording

What are the classes of information necessary for debugging?

<table>
<thead>
<tr>
<th>Coredump</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101010</td>
<td>11101011</td>
</tr>
<tr>
<td>10101011</td>
<td>10001001</td>
</tr>
</tbody>
</table>
Debug Without Recording

Coredump
10101010
10101011

Program
11101011
10001001

What are the classes of information necessary for debugging?

Synthesize

program inputs
thread schedule
Debug Without Recording

What are the classes of information necessary for debugging?

Coredump
10101010
10101011

Program
11101011
10001001

Synthesize

program inputs
thread schedule

Original Program
Binary

Replay Library

Debugger
Debug Without Recording

Synthesize

program inputs

thread schedule
Debug Without Recording

Synthesize

program inputs

thread schedule
Debug Without Recording

Synthesize

program inputs

thread schedule
Debug Without Recording

Synthesize

*program inputs*

*thread schedule*
Debug Without Recording

Synthesize

*program inputs*

*thread schedule*
Debug Without Recording

Synthesize

program inputs

thread schedule
Debug Without Recording

Synthesize

program inputs

thread schedule

Exact same execution is not necessary
ODR, PRES (SOSP’09) and ESD (EuroSys’10)
Debug Without Recording

Synthesize

- program inputs
- thread schedule

Exact same execution is not necessary
ODR, PRES (SOSP’09) and ESD (EuroSys’10)

Reproduce the root cause and the failure
Debug Determinism (HotOS’11)
Debug Without Recording

Synthesize

- program inputs
- thread schedule

Exact same execution is not necessary
ODR, PRES (SOSP’09) and ESD (EuroSys’10)

Reproduce the root cause and the failure
Debug Determinism (HotOS’11)
Reverse Execution Synthesis

Synthesize

program inputs

thread schedule
Reverse Execution Synthesis

Synthesize

*program inputs*

*thread schedule*

The root cause is close to the failure 85% of the time (Conseq, ASPLOS’11)
Reverse Execution Synthesis

Synthesize

program inputs

thread schedule

The root cause is close to the failure 85% of the time (Conseq, ASPLOS’11)
Reverse Execution Synthesis

Synthesize

*program inputs*

*thread schedule*

The root cause is close to the failure 85% of the time (Conseq, ASPLOS’11)
Reverse Execution Synthesis

Synthesize

*program inputs*

*thread schedule*

The root cause is close to the failure 85% of the time (Conseq, ASPLOS’11)
Reverse Execution Synthesis
Reverse Execution Synthesis

```c
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
    buffer[y] = 1
```
Reverse Execution Synthesis

```c
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
    buffer[y] = 1
```

Coredump:

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>
Reverse Execution Synthesis

```plaintext
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
    buffer[y] = 1
```

```
buffer[y] = 1;

<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>
```

Coredump: (buffer overflow)
Reverse Execution Synthesis

\[
x = 1;
if \ (f(x) == y) \{
\text{goto next;}\}
\]

\[
\ldots
\]

\[
x = 2;
if \ (g(x) == y) \{
\text{goto next;}\}
exit();
\]

\text{next:}
\text{buffer}[y] = 1

\textbf{Coredump:} (buffer overflow)

\[
\begin{array}{c|c}
  x & 1 \\
  y & 10 \\
\end{array}
\]
Reverse Execution Synthesis

\[
x = 1;
if \ (f(x) == y) \{
  goto \ next;
}\]

\[
...\]

\[
x = 2;
if \ (g(x) == y) \{
  goto \ next;
}\exit();\]

next:
buffer[y] = 1

Coredump: (buffer overflow)

\[
\begin{array}{c|c}
  x & 1 \\
  y & 10 \\
\end{array}
\]
Reverse Execution Synthesis

\[
\begin{align*}
x &= 1; \\
&\text{if } \ f(x) == y) \\
&\quad \text{goto next;}
\end{align*}
\]

... 
\[
\begin{align*}
x &= 2; \\
&\text{if } \ g(x) == y) \\
&\quad \text{goto next;}
\end{align*}
\]

exit();

\text{next:}
\[
\begin{array}{c}
\text{buffer[y] = 1}
\end{array}
\]

\text{Coredump: (buffer overflow)}

\[
\begin{array}{c|c}
\text{x} & \text{1} \\
\hline
\text{y} & \text{10}
\end{array}
\]
Reverse Execution Synthesis

x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
buffer[y] = 1

Coredump: (buffer overflow)

x  1
y  10
Reverse Execution Synthesis

\[
\begin{array}{c}
x = 1; \\
\text{if } (f(x) == y) \\
\text{goto next;}
\end{array}
\]

\[
\begin{array}{c}
\text{...} \\
x = 2; \\
\text{if } (g(x) == y) \\
\text{goto next;}
\end{array}
\]

\[
\begin{array}{c}
\text{exit();} \\
\text{next:} \\
\text{buffer[y] = 1}
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{x} & \text{y} & \text{x} & \text{y} & \text{x} & \text{y} \\
\hline
1 & 10 & 2 & 10 & 1 & 10 \\
\end{array}
\]

Coredump: (buffer overflow)
Reverse Execution Synthesis

```
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();

next:
buffer[y] = 1
```

```
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
```

Coredump: (buffer overflow)
Reverse Execution Synthesis

\[
\begin{align*}
x &= 1; \\
\text{if } (f(x) == y) &\{ \\
\text{goto next}; \\
\}
\end{align*}
\]

\[
\begin{align*}
x &= 2; \\
\text{if } (g(x) == y) &\{ \\
\text{goto next}; \\
\}
\end{align*}
\]

\[
\text{exit();}
\text{next:}
\]

\[
\text{buffer}[y] = 1
\]

Coredump: (buffer overflow)

\[
\begin{array}{c|c|c}
\hline
x & 1 \\
\hline
y & 10 \\
\hline
\end{array}
\]
Reverse Execution Synthesis

Coredump: (buffer overflow)

x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();

next:
buffer[y] = 1;
Reverse Execution Synthesis

```
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:  
buffer[y] = 1
```

---

Diagram:

```
  x
  ┌── y
  │  └── x
  │     ┌── if (f(x) == y)
  │     │
  │     └── True
  │          ┌── buffer[y] = 1;
  │          │
  │          ├── Coredump:
  │          │  (buffer overflow)
  │          │      ┌── x
  │          │      │  1
  │          │      │      ┌── y
  │          │      │      │  10
  │          │      │      └── g(2) != 10
  │          │      └── False
  │               └── False
  │                   └── True
  └─ x
    ┌── y
    │
    └── x
```
x = 1;
if (f(x) == y) {
  goto next;
}
...
x = 2;
if (g(x) == y) {
  goto next;
}
exit();

buffer[y] = 1

Reverse Execution Synthesis

x
---
y

x = 1;
if (f(x) == y)  
  x = 2;
if (g(x) == y)

True
False

True
False

buffer[y] = 1;

Coredump: (buffer overflow)

x 1
---
y 10

x 2
---
y 10

g(2) != 10
Reverse Execution Synthesis

```
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
buffer[y] = 1
```

![Decision tree diagram](image)

```
x = 1;  
if (f(x) == y)  
    x = 2;  
if (g(x) == y)  
    buffer[y] = 1

Coredump: (buffer overflow)
```

```
x 1  
y 10
```

```
x 2  
y 10
```

```g(2) != 10```

no match
Reverse Execution Synthesis

\[
\begin{array}{c}
x = 1; \\
\text{if } (f(x) == y) \\
\text{goto next;}
\end{array}
\]

\[
\begin{array}{c}
\ldots \\
x = 2; \\
\text{if } (g(x) == y) \\
\text{goto next;}
\end{array}
\]

\[\text{exit();} \]

\[
\text{next: buffer[y] = 1}
\]

**Coredump:** (buffer overflow)

\[
\begin{array}{c}
\text{x} \\
\text{1} \\
\text{y} \\
\text{10}
\end{array}
\]

\[
\begin{array}{c}
\text{x} \\
\text{2} \\
\text{y} \\
\text{10}
\end{array}
\]

\[
g(2) != 10
\]
Reverse Execution Synthesis

\[
\begin{array}{c|c|c}
\text{x} & \text{y} & \text{x} \quad \text{?} \\
\hline
\text{y} & \text{y} & \text{y} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{x} & \text{y} & \text{buffer[y] = 1;}
\end{array}
\]

Coredump: (buffer overflow)

- \[
\begin{array}{c|c|c}
\text{x} & \text{y} & \text{y} \\
\hline
1 & 10 & 10
\end{array}
\]

- \[
\begin{array}{c|c|c}
\text{x} & \text{y} & \text{y} \\
\hline
2 & 10 & 10
\end{array}
\]
Reverse Execution Synthesis

\[
\begin{align*}
\text{if } (f(x) == y) & \text{ goto next; } \\
\text{if } (g(x) == y) & \text{ exit(); } \\
\text{buffer}[y] & = 1 \text{ (buffer overflow)}
\end{align*}
\]
Reverse Execution Synthesis

```
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
```

```
x = 1;
if (f(x) == y)
    x = 2;
if (g(x) == y)
    buffer[y] = 1;  # buffer overflow
```

```
x 1
y

x = 1;
if (f(x) == y)
    buffer[y] = 1;

x 2
? y 10

x = 2;
if (g(x) == y)
    g(2) != 10
```

```
x 1
y 10

False

True

buffer[y] = 1;
```

```
x 2
y 10

True

False

Coredump:
(buffer overflow)

x 1
y 10

x 2
y 10

g(2) != 10
```
Reverse Execution Synthesis

\[
\begin{align*}
\text{if } (f(x) == y) \{ & \quad \text{goto next;} \\
\text{if } (g(x) == y) \{ & \quad \text{goto next;} \\
\text{exit();} \\
\text{next:} \quad & \quad \text{buffer}[y] = 1
\end{align*}
\]

Coredump: (buffer overflow)

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]

\[
\begin{align*}
x & = 1; \\
\text{if } (f(x) == y) \{ & \quad x = 2; \\
\text{if } (g(x) == y) \{ & \quad \text{buffer}[y] = 1
\end{align*}
\]
x = 1;
if (f(x) == y) {
  goto next;
}
...
x = 2;
if (g(x) == y) {
  goto next;
}
ext();

next:
buffer[y] = 1
Reverse Execution Synthesis

```
x = 1;
if (f(x) == y) {
  goto next;
}
...
x = 2;
if (g(x) == y) {
  goto next;
}
exit();
next:
buffer[y] = 1;
```

**Coredump:** (buffer overflow)

True  False
---  ---
True
False

x = 1;
if (f(x) == y)

x = 2;
if (g(x) == y)

buffer[y] = 1;
Reverse Execution Synthesis

\[
\begin{array}{c|cc|c|c}
& x & ? & x & ? \\
\hline
\text{False} & y & 10 & y & 10 \\
\text{True} & & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
& x & 1 & x & 1 & x & 2 \\
\hline
\text{False} & \text{buffer}[y] = 1; & & & & \\
\text{True} & & & & & \\
\end{array}
\]

Coredump: (buffer overflow)

\[
\begin{array}{c|c|c|c|c}
& x & y & 10 & y & 10 & y & 10 \\
\hline
\text{False} & f(1) == 10 & & & & & \\
\text{True} & & & & & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
& x & y & 10 & y & 10 \\
\hline
\text{False} & g(2) != 10 & & & & \\
\text{True} & & & & & \\
\end{array}
\]
Reverse Execution Synthesis

\[
\begin{array}{c|c|c}
\text{x} & \text{?} & \text{y} \\
\hline
1 & 10 & \text{10}
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{x} & \text{?} & \text{y} \\
\hline
2 & 10 & \text{10}
\end{array}
\]

\[
\begin{array}{c|c}
\text{x} & \text{1} \\
\hline
\text{y} & \text{10}
\end{array}
\]

\[
\begin{array}{c|c}
\text{x} & \text{1} \\
\hline
\text{y} & \text{10}
\end{array}
\]

\[
\begin{array}{c|c}
\text{x} & \text{2} \\
\hline
\text{y} & \text{10}
\end{array}
\]

\[
\begin{array}{c|c}
\text{x} & \text{2} \\
\hline
\text{y} & \text{10}
\end{array}
\]

match
Reverse Execution Synthesis

Coredump: (buffer overflow)

match
Reverse Execution Synthesis

```
x = 1;
if (f(x) == y) {
    goto next;
}
...
x = 2;
if (g(x) == y) {
    goto next;
}
exit();
next:
buffer[y] = 1
```

Execution suffix

<table>
<thead>
<tr>
<th>x</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>

```
x = 1;
if (f(x) == y) {
    x = 2;
    if (g(x) == y)
}
```

```
x = 2;
if (g(x) == y) {
    buffer[y] = 1
```

Coredump: (buffer overflow)

<table>
<thead>
<tr>
<th>x</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>

f(1) == 10

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>

match

<table>
<thead>
<tr>
<th>x</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>10</td>
</tr>
</tbody>
</table>

g(2) != 10

(gdb) reverse-step
Reverse Execution Synthesis

- Coredump + program $\rightarrow$ execution suffix
- Debug arbitrarily long executions
- No runtime recording
Use Cases

• Automated debugging
  • *identify the root cause of a failure*

• Automated bug triaging
  • *trriage based on the execution suffix*

• Identify likely hardware errors
  • *when no execution suffix explains the coredump*