Fault Localization with CPAchecker

Thomas Lemberger

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CPA

SoSy-Lab Software Systems

Debugging

3 steps of debugging:

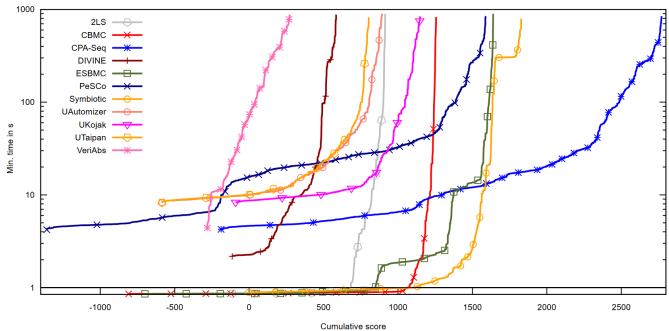
1. Recognize failure
2. Localize fault
3. Fix fault

Debugging

3 steps of debugging:

1. Recognize failure
2. Localize fault
3. Fix fault

- CPAchecker is really good at first step



SV-COMP '20 falsification scores

Debugging

3 steps of debugging:

1. Recognize failure
2. Localize fault
3. Fix fault

- CPAchecker is really good at first step
- Our goal: Support second step

Fault Localization

- Lots of existing research
  - Test-based techniques
  - Formal-verification-based techniques
- At the moment, we adapt existing approaches (ongoing work)
- Existing approaches based on three ideas:
  - Coverage analysis[1][2][3]
  - Path comparison[4][5][6]
  - Error-trace analysis[7][8]

[1] W. E. Wong et al. "The DStar Method for Effective Software Fault Localization". In: *IEEE Trans. Reliab.* 63.1 (2014).

[2] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. "On the Accuracy of Spectrum-based Fault Localization". In: *Proc. TAICPART*. 2007.

[3] J. A. Jones, M. J. Harrold, and J. T. Stasko. "Visualization of Test Information to Assist Fault Localization". In: *Proc. ICSE*. 2002.

[4] L. Guo, A. Roychoudhury, and T. Wang. "Accurately Choosing Execution Runs for Software Fault Localization". In: *Proc. CC*. 2006.

[5] S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: *Proc. FSE*. 2004.

[6] T. Wang and A. Roychoudhury. "Automated Path Generation for Software Fault Localization". In: *Proc. ASE*. 2005.

[7] E. Ermis, M. Schäf, and T. Wies. "Error Invariants". In: *Proc. FM*. LNCS 7436. Springer, 2012.

[8] M. Jose and R. Majumdar. "Cause Clue Clauses: Error Localization using Maximum Satisfiability". In: *Proc. PLDI*. ACM, 2011.

Next: 3 selected techniques

1. Tarantula
2. Abstract Distance Metric
3. Error Invariants

1. Tarantula[9] (implemented by Schindar Ali)

- Uses statement coverage of error paths and safe program paths
- Computes *suspiciousness* for each code statement s

$$\texttt{suspiciousness(s)} = \frac{\left(\frac{\texttt{errorPaths(s)}}{\texttt{allErrorPaths}}\right)}{\left(\frac{\texttt{errorPaths(s)}}{\texttt{allErrorPaths}}\right) + \left(\frac{\texttt{safePaths(s)}}{\texttt{allSafePaths}}\right)}$$

- Ranking from suspiciousness

[9] J. A. Jones, M. J. Harrold, and J. T. Stasko. "Visualization of Test Information to Assist Fault Localization". In: *Proc. ICSE*. 2002.

1. Tarantula

$$\texttt{suspiciousness(s)} = \frac{\left(\frac{\texttt{errorPaths(s)}}{\texttt{allErrorPaths}}\right)}{\left(\frac{\texttt{errorPaths(s)}}{\texttt{allErrorPaths}}\right) + \left(\frac{\texttt{safePaths(s)}}{\texttt{allSafePaths}}\right)}$$

		Test Cases							
	`mid() { int x,y,z,m;`	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	suspiciousness	rank
1:	`read("Enter 3 numbers:",x,y,z);`	●	●	●	●	●	●	0.5	7
2:	`m = z;`	●	●	●	●	●	●	0.5	7
3:	`if (y<z)`	●	●	●	●	●	●	0.5	7
4:	`if (x<y)`	●	●			●	●	0.63	3
5:	`m = y;`		●					0.0	13
6:	`else if (x<z)`	●				●	●	0.71	2
7:	`m = y; // *** bug ***`	●					●	0.83	1
8:	`else`			●	●			0.0	13
9:	`if (x>y)`			●	●			0.0	13
10:	`m = y;`			●				0.0	13
11:	`else if (x>z)`				●			0.0	13
12:	`m = x;`							0.0	13
13:	`print("Middle number is:",m);`	●	●	●	●	●	●	0.5	7
	`}` Pass/Fail Status	P	P	P	P	P	F		

Example from J. A. Jones and M. J. Harrold. "Empirical evaluation of the tarantula automatic fault-localization technique". In: *Proc. ASE*. 2005

1. Tarantula

$$\texttt{suspiciousness(s)} = \frac{\left(\frac{\texttt{errorPaths(s)}}{\texttt{allErrorPaths}}\right)}{\left(\frac{\texttt{errorPaths(s)}}{\texttt{allErrorPaths}}\right)+\left(\frac{\texttt{safePaths(s)}}{\texttt{allSafePaths}}\right)}$$

		Test Cases							
	mid() { int x,y,z,m;	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	suspiciousness	rank
1:	read("Enter 3 numbers:",x,y,z);	●	●	●	●	●	●	0.5	7
2:	m = z;	●	●	●	●	●	●	0.5	7
3:	if (y<z)	●	●	●	●	●	●	0.5	7
4:	if (x<y)	●	●			●	●	0.63	3
5:	m = y;		●					0.0	13
6:	else if (x<z)	●				●	●	0.71	2
7:	m = y; // *** bug ***	●					●	0.83	1
8:	else			●	●			0.0	13
9:	if (x>y)			●	●			0.0	13
10:	m = y;			●				0.0	13
11:	else if (x>z)				●			0.0	13
12:	m = x;							0.0	13
13:	print("Middle number is:",m);	●	●	●	●	●	●	0.5	7
	} Pass/Fail Status	P	P	P	P	P	F		

- Works best on precise analyses
- Requires distinct paths
- → Open question: How to handle abstract paths?

Example from J. A. Jones and M. J. Harrold. "Empirical evaluation of the tarantula automatic fault-localization technique". In: *Proc. ASE*. 2005

2. Abstract Distance Metric[10] (implemented by Angelos Kafounis)

1. Consider single error trace
2. Compare all safe paths with error trace
3. Find safe path with smallest *distance* to error trace

→ Code difference is fault

[10] S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: *Proc. FSE*. 2004.

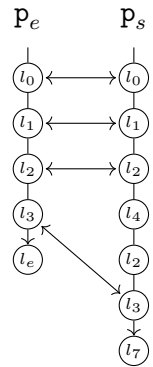
2. Abstract Distance Metric[10] (implemented by Angelos Kafounis)

1. Consider single error trace
2. Compare all safe paths with error trace
3. Find safe path with smallest *distance* to error trace

→ Code difference is fault

Considered distances:

- Syntactic path (alignment and unalignment)



$$\text{align}(\mathtt{p}_e, \mathtt{p}_s) = 4$$
$$\text{unalign}(\mathtt{p}_e, \mathtt{p}_s) = 1 + 3$$

[10] S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: *Proc. FSE*. 2004.

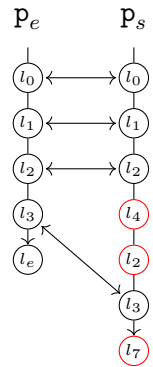
2. Abstract Distance Metric[10] (implemented by Angelos Kafounis)

1. Consider single error trace
2. Compare all safe paths with error trace
3. Find safe path with smallest *distance* to error trace

→ Code difference is fault

Considered distances:

- Syntactic path (alignment and unalignment)



$$\text{align}(\mathtt{p}_e, \mathtt{p}_s) = 4$$
$$\text{unalign}(\mathtt{p}_e, \mathtt{p}_s) = 1 + 3$$

[10] S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: *Proc. FSE*. 2004.

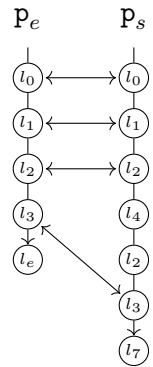
2. Abstract Distance Metric[10] (implemented by Angelos Kafounis)

1. Consider single error trace
2. Compare all safe paths with error trace
3. Find safe path with smallest *distance* to error trace

$\rightarrow$ Code difference is fault

Considered distances:

- Syntactic path (alignment and unalignment)
- Predicates in aligned abstract states



$$\mathrm{align}(\mathtt{p}_e, \mathtt{p}_s) = 4$$
$$\mathrm{unalign}(\mathtt{p}_e, \mathtt{p}_s) = 1 + 3$$

[10] S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: *Proc. FSE*. 2004.

3. Error Invariants[11] (implemented by Matthias Kettl)

1. Consider single error trace
2. Build formula for error trace
3. Compute inductive Craig interpolants
4. Change in interpolants: Fault *contribution*

[11] E. Ermis, M. Schäf, and T. Wies. "Error Invariants". In: *Proc. FM*. LNCS 7436. Springer, 2012.

Error Invariants: Build Trace Formula

- We assume the last if-condition before an error is the post-condition ϕ

```
if (x <= 0)
reach_error();
```

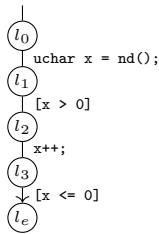
$\Rightarrow \phi = x > 0$

Error Invariants: Build Trace Formula

- We assume the last if-condition before an error is the post-condition ϕ

1. Consider single error trace

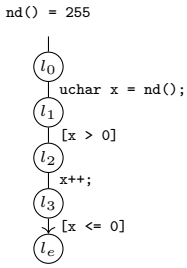
$$\mathrm{p}_e = l_0 \xrightarrow{op_0} l_1 \ldots \xrightarrow{op_{n-2}} l_{n-1} \xrightarrow{\neg\phi} l_e$$



Error Invariants: Build Trace Formula

- We assume the last if-condition before an error is the post-condition ϕ

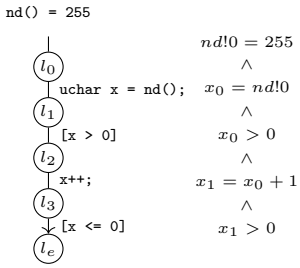
1. Consider single error trace
   $\mathrm{p}_e = l_0 \xrightarrow{op_0} l_1 \ldots \xrightarrow{op_{n-2}} l_{n-1} \xrightarrow{\neg\phi} l_e$
2. Compute program inputs that produce p_e



Error Invariants: Build Trace Formula

- We assume the last if-condition before an error is the post-condition ϕ

1. Consider single error trace
   $\mathrm{p}_e = l_0 \xrightarrow{op_0} l_1 \ldots \xrightarrow{op_{n-2}} l_{n-1} \xrightarrow{\neg\phi} l_e$
2. Compute program inputs that produce p_e
3. Build trace formula $\mathrm{inputs} \wedge \mathtt{TF}(\mathrm{p}_e[0, n-2]) \wedge \phi$
   $\Rightarrow$ This formula is UNSAT



Error Invariants: Craig Interpolation

- Given formulas A (*prefix*) and B (*suffix*) with $A \wedge B$ unsat.
- Formula I is a *Craig interpolant* for A and B, if:
  1. $A \Rightarrow I$
  2. $I \wedge B$ unsat
  3. I only consists of variables that occur in both A and B

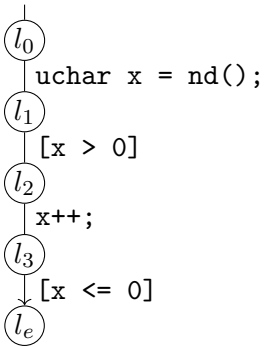
Error Invariants: Craig Interpolation

- We compute *inductive* Craig interpolants I_0 to I_{n-2} for each step in $\mathtt{p}_e$:
  - I_0 Craig interpolant for $A = \text{inputs} \wedge \mathtt{TF}(\mathtt{p}_e[0])$ and $B = \mathtt{TF}(\mathtt{p}_e[1, n-2]) \wedge \phi$
  - I_i Craig interpolant for $A = I_{i-1} \wedge \mathtt{TF}(\mathtt{p}_e[i])$ and $B = \mathtt{TF}(\mathtt{p}_e[i+1, n-2]) \wedge \phi$
- Result: Abstract error trace $true \xrightarrow{op_0} I_0 \ldots \xrightarrow{op_{n-2}} I_{n-2} \xrightarrow{\phi} l_e$
- Idea: Interpolants explain what information is relevant to error $\neg\phi$

  $\Rightarrow$ Interpolant changes from $I_{i-1} \xrightarrow{op_i} I_i$? Then op_i is relevant.

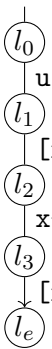
Error Invariants: Example

$\mathtt{p}_e$



Error Invariants: Example

$\mathtt{p}_e$

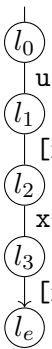


Trace formula

$$
\begin{array}{c}
nd!0 = 255 \\
\wedge \\
x_0 = nd!0 \\
\wedge \\
x_0 > 0 \\
\wedge \\
x_1 = x_0 + 1 \\
\wedge \\
x_1 > 0
\end{array}
$$

Error Invariants: Example

$\mathtt{P}_e$



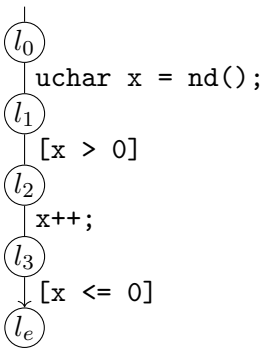
Trace formula

$$nd!0 = 255 \wedge x_0 = nd!0 \wedge \underbrace{x_0 > 0}_{I_0} \wedge x_0 = 255 \wedge \underbrace{x_1 = x_0 + 1}_{I_1} \wedge x_1 = 0 \wedge \underbrace{x_1 > 0}_{I_2}$$

Interpolants

Error Invariants: Example

$\mathtt{p}_e$

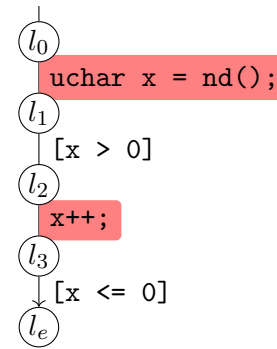


Trace formula

Interpolants

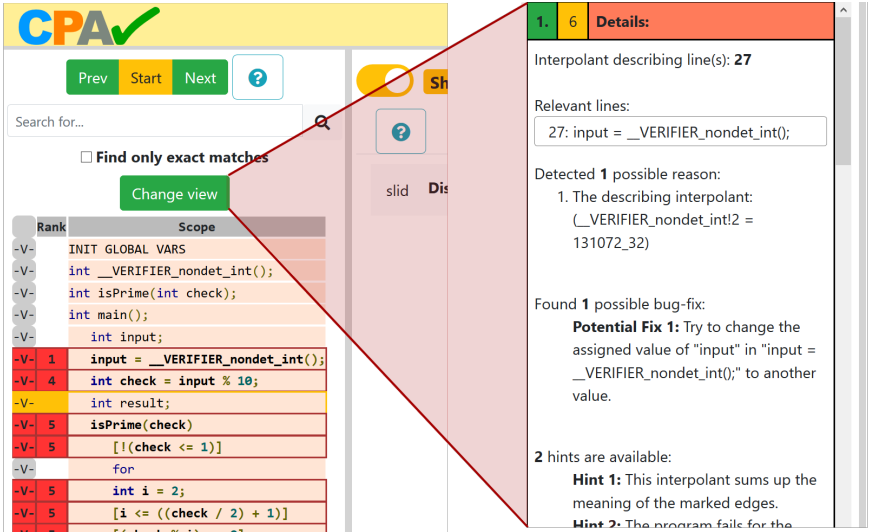
$$
\begin{array}{l}
nd!0 = 255 \\
\quad \wedge \\
x_0 = nd!0 \\
\quad \wedge \overline{\qquad I_0 \qquad} \; x_0 = 255 \\
x_0 > 0 \\
\quad \wedge \overline{\qquad I_1 \qquad} \; x_0 = 255 \\
x_1 = x_0 + 1 \\
\quad \wedge \overline{\qquad I_2 \qquad} \; x_1 = 0 \\
x_1 > 0
\end{array}
$$

Fault contributions



Visualization of Localization Results (implemented by Matthias Kettl)

- Localization info included in Report.html



Future Work

- Perform large evaluation
- Improve existing techniques

Conclusion

- CPAchecker can now locate faults!
- 3 fundamentally different groups of techniques:
  - Coverage analysis
  - Path comparison
  - Error-trace analysis