



Fault Localization with CPAchecker

Thomas Lemberger



CPA✓

A large, stylized logo for CPAchecker. The letters 'CPA' are in a bold, sans-serif font, with 'CPA' in blue and 'A' in grey. A large green checkmark is positioned to the right of the 'A'. The entire logo is set against a white background.

SoSy-Lab
Software Systems

A blue rounded rectangle containing the text 'SoSy-Lab' in white, with 'Software Systems' in smaller text below it.

Debugging

3 steps of debugging:

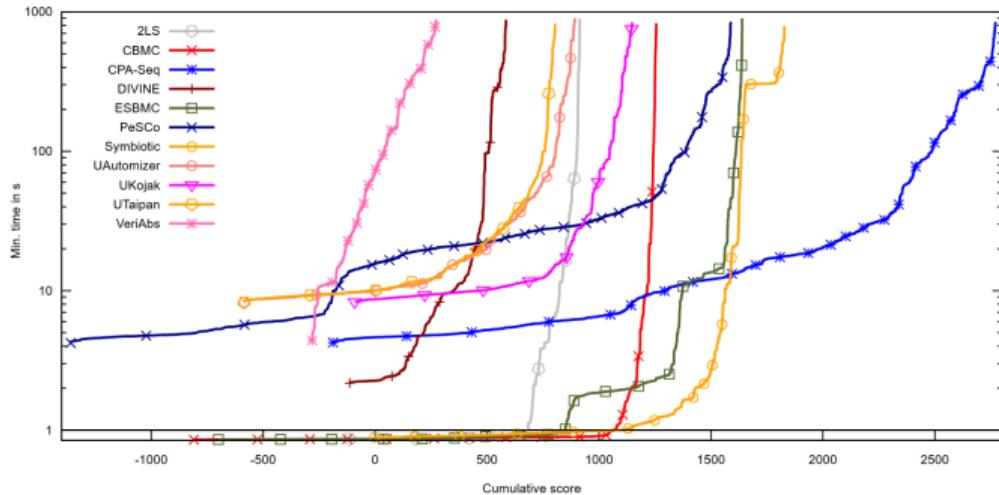
1. Recognize failure
2. Localize fault
3. Fix fault

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
 - ▶ CPAchecker is really good at first step
2. Localize fault
 - ▶ Our goal: Support second step
3. Fix fault

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 analysis¹²³
 - ▶ Path comparison⁴⁵⁶
 - ▶ Error-trace analysis⁷⁸

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

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

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

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

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

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

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

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

Next: 3 selected techniques

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

1. Tarantula⁹ (implemented by Schindar Ali)

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

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

- ▶ Ranking from suspiciousness

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

1. Tarantula

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

		Test Cases						suspiciousness	rank
		3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3		
mid()	{								
	int x,y,z,m;								
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

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

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

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

2. Abstract Distance Metric¹⁰(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

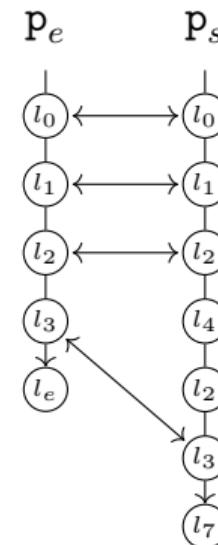
¹⁰S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: Proc. FSE. 2004.
Thomas Lemberger

2. Abstract Distance Metric¹⁰(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)



$$\text{align}(p_e, p_s) = 4$$

$$\text{unalign}(p_e, p_s) = 1 + 3$$

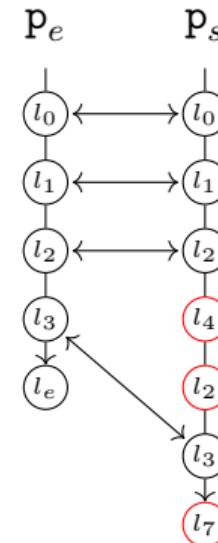
¹⁰S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: Proc. FSE. 2004.
Thomas Lemberger

2. Abstract Distance Metric¹⁰(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)



$$\text{align}(p_e, p_s) = 4$$

$$\text{unalign}(p_e, p_s) = 1 + 3$$

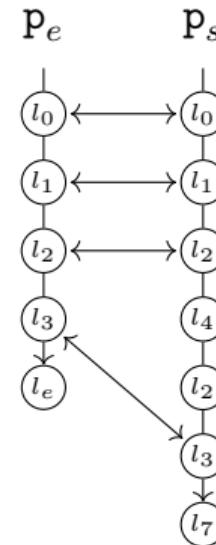
¹⁰S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: Proc. FSE. 2004.
Thomas Lemberger

2. Abstract Distance Metric¹⁰(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)
- ▶ Predicates in aligned abstract states



$$\text{align}(p_e, p_s) = 4$$

$$\text{unalign}(p_e, p_s) = 1 + 3$$

¹⁰S. Chaki, A. Groce, and O. Strichman. "Explaining Abstract Counterexamples". In: Proc. FSE. 2004.
Thomas Lemberger

3. Error Invariants¹¹ (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*

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

Error Invariants: Build Trace Formula

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

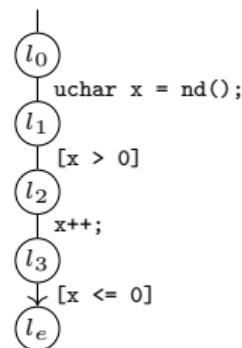
```
if (x <= 0)
reach_error();
⇒ φ = x > 0
```

Error Invariants: Build Trace Formula

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

1. Consider single error trace

$$p_e = l_0 \xrightarrow{op_0} l_1 \dots \xrightarrow{op_{n-2}} l_{n-1} \xrightarrow{\neg\phi} l_e$$



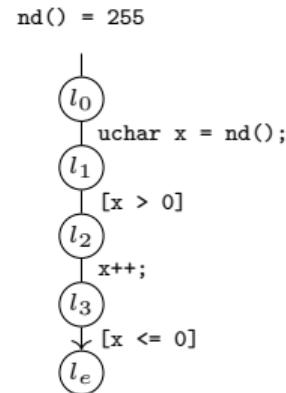
Error Invariants: Build Trace Formula

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

1. Consider single error trace

$$p_e = l_0 \xrightarrow{op_0} l_1 \dots \xrightarrow{op_{n-2}} l_{n-1} \xrightarrow{\neg\phi} l_e$$

2. Compute program inputs that produce p_e



Error Invariants: Build Trace Formula

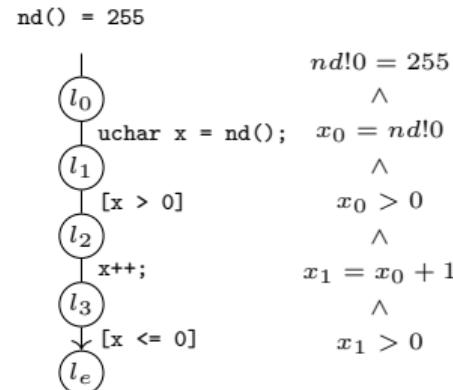
- We assume the last if-condition before an error is the post-condition ϕ

1. Consider single error trace

$$p_e = l_0 \xrightarrow{op_0} l_1 \dots \xrightarrow{op_{n-2}} l_{n-1} \xrightarrow{\neg\phi} l_e$$

2. Compute program inputs that produce p_e

3. Build trace formula inputs $\wedge \text{TF}(p_e[0, n-2]) \wedge \phi$
⇒ This formula is UNSAT



Error Invariants: Craig Interpolation

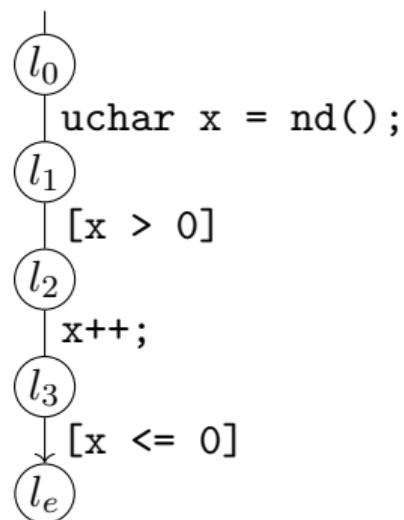
- ▶ Given formulas A (*prefix*) and B (*suffix*) with $A \wedge B$ unsat.
- ▶ Formula I is a *Craig interpolant* for A and B , if:
 1. $A \Rightarrow I$
 2. $I \wedge B$ unsat
 3. I only consists of variables that occur in both A and B

Error Invariants: Craig Interpolation

- ▶ We compute *inductive* Craig interpolants I_0 to I_{n-2} for each step in p_e :
 - I_0 Craig interpolant for $A = \text{inputs} \wedge \text{TF}(p_e[0])$ and $B = \text{TF}(p_e[1, n-2]) \wedge \phi$
 - I_i Craig interpolant for $A = I_{i-1} \wedge \text{TF}(p_e[i])$ and $B = \text{TF}(p_e[i+1, n-2]) \wedge \phi$
- ▶ Result: Abstract error trace $true \xrightarrow{op_0} I_0 \dots \xrightarrow{op_{n-2}} I_{n-2} \xrightarrow{\phi} l_e$
- ▶ 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

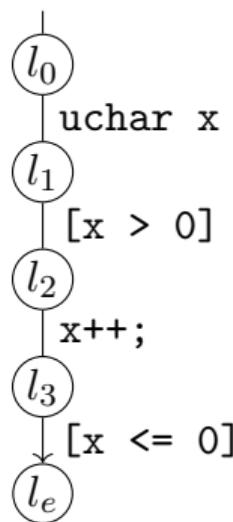
p_e



Error Invariants: Example

p_e

Trace
formula



$$nd!0 = 255$$

\wedge

$$x_0 = nd!0$$

\wedge

$$x_0 > 0$$

\wedge

$$x_1 = x_0 + 1$$

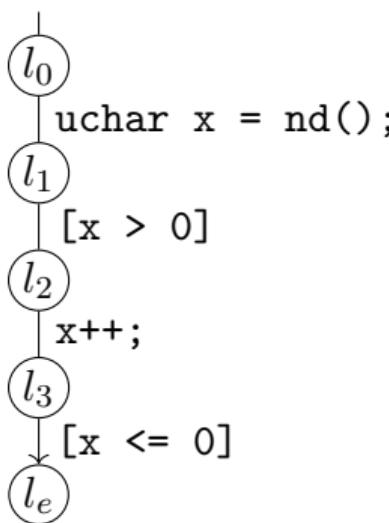
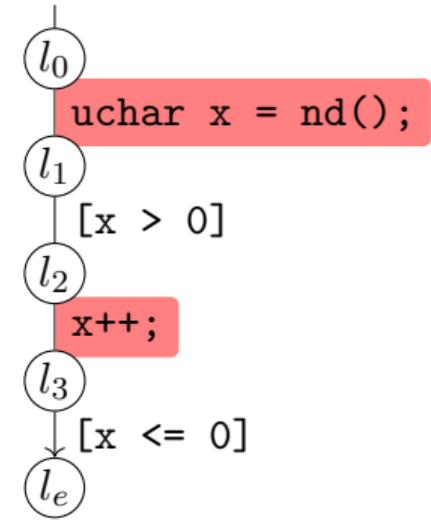
\wedge

$$x_1 > 0$$

Error Invariants: Example

p_e	Trace formula	Interpolants
l_0	$nd!0 = 255$	
l_1	\wedge	
uchar x = nd();	$x_0 = nd!0$	I_0
l_2	\wedge	$x_0 = 255$
[x > 0]	$x_0 > 0$	I_1
l_3	\wedge	$x_0 = 255$
x++;	$x_1 = x_0 + 1$	I_2
l_e	\wedge	$x_1 = 0$
[x <= 0]	$x_1 > 0$	

Error Invariants: Example

p_e	Trace formula	Interpolants	Fault contributions
 <pre>graph TD; l0((l0)) -- "uchar x = nd();" --> l1((l1)); l1 -- "[x > 0]" --> l2((l2)); l2 -- "x++;" --> l3((l3)); l3 -- "[x <= 0]" --> le((le))</pre>	$\begin{aligned} nd!0 &= 255 \\ \wedge \\ x_0 &= nd!0 \quad I_0 \\ \wedge \quad & x_0 = 255 \\ x_0 &> 0 \quad I_1 \\ \wedge \quad & x_0 = 255 \\ x_1 &= x_0 + 1 \quad I_2 \\ \wedge \quad & x_1 = 0 \\ x_1 &> 0 \end{aligned}$		 <pre>graph TD; l0((l0)) --- l1((l1)); l1 --- l2((l2)); l2 --- l3((l3)); l3 --- le((le))</pre> <p>uchar x = nd(); [x > 0] x++; [x <= 0]</p>

Visualization of Localization Results (implemented by Matthias Kettl)

- ▶ Localization info included in Report.html

The screenshot shows the CPA tool interface with a sidebar and a main analysis panel.

Left Sidebar:

- CPA logo with a green checkmark.
- Navigation buttons: Prev, Start, Next, and a question mark icon.
- Search bar: Search for...
- Checkboxes: Find only exact matches and Change view.
- Table header: Rank, Scope.
- Table rows (highlighted in red for row 1):
 - V- INIT GLOBAL VARS
 - V- int __VERIFIER_nondet_int();
 - V- int isPrime(int check);
 - V- int main();
 - V- int input;
 - V- 1** input = __VERIFIER_nondet_int();
 - V- 4 int check = input % 10;
 - V- int result;
 - V- 5 isPrime(check)
 - V- 5 [!(check <= 1)]
 - V- for
 - V- 5 int i = 2;
 - V- 5 [i <= ((check / 2) + 1)]
 - V- 5 [(check % i) == 0]
 - V- 6 Thomas Lemburger return 0;

Right Panel:

- Details:** 1. 6 Details:
 - Interpolant describing line(s): **27**
 - Relevant lines:
27: input = __VERIFIER_nondet_int();
 - Detected 1 possible reason:
 1. The describing interpolant:
(__VERIFIER_nondet_int!2 = 131072_32)
 - Found 1 possible bug-fix:
Potential Fix 1: Try to change the assigned value of "input" in "input = __VERIFIER_nondet_int();" to another value.
 - 2 hints are available:
 - Hint 1:** This interpolant sums up the meaning of the marked edges.
 - Hint 2:** The program fails for the

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

