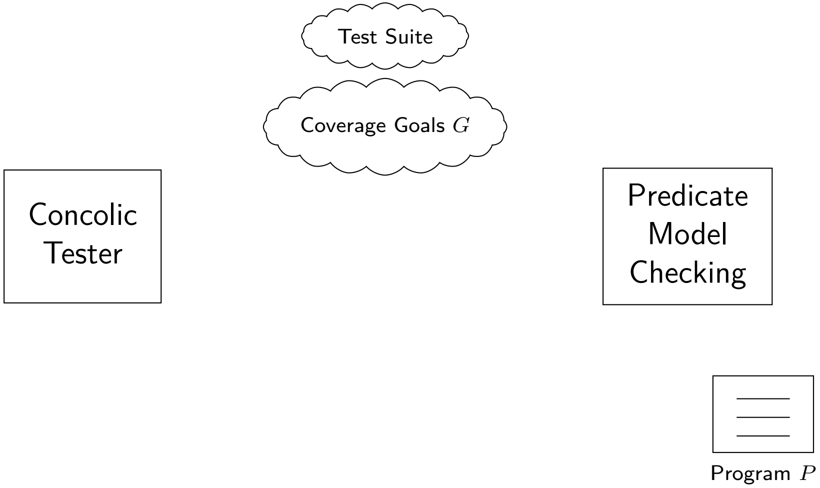
Cooperative Test-Case Generation

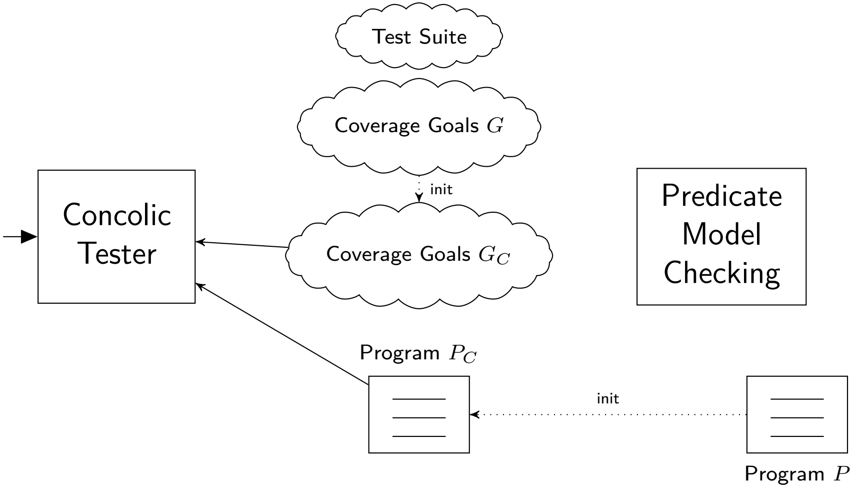
Marie-Christine Jakobs

LMU Munich, Germany

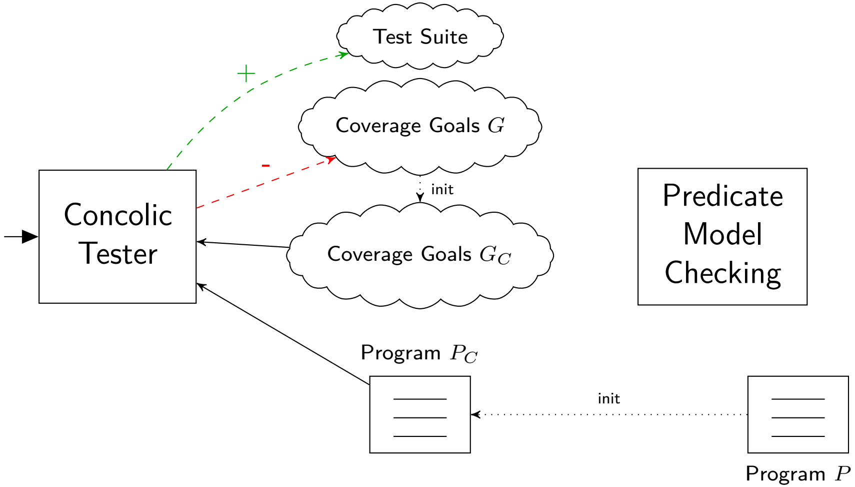
Abstraction-driven Concolic Testing
[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]



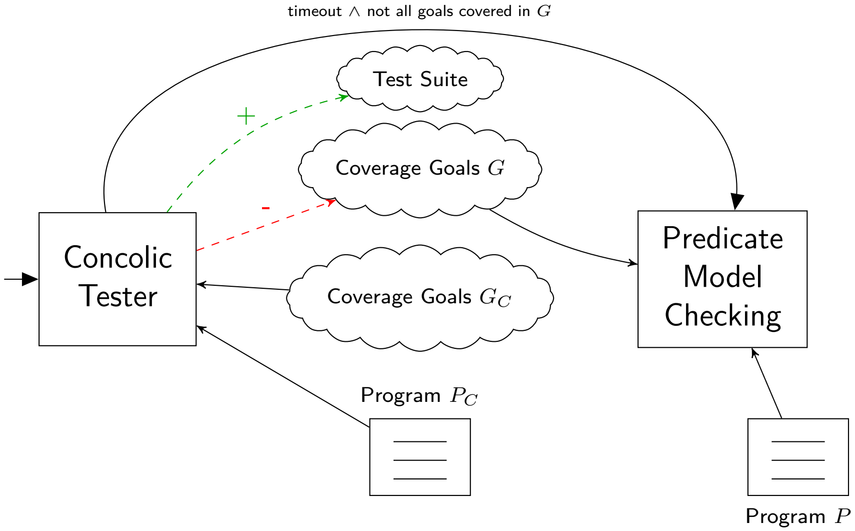
Abstraction-driven Concolic Testing
[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]



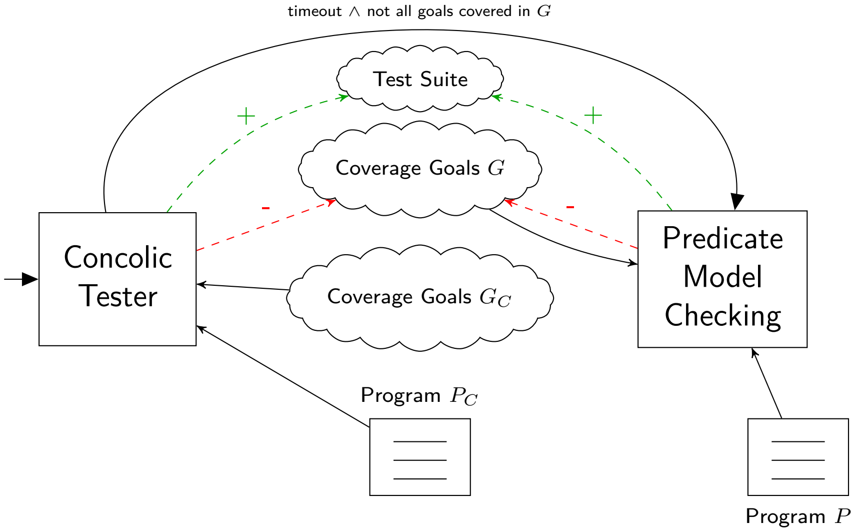
Abstraction-driven Concolic Testing
[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]



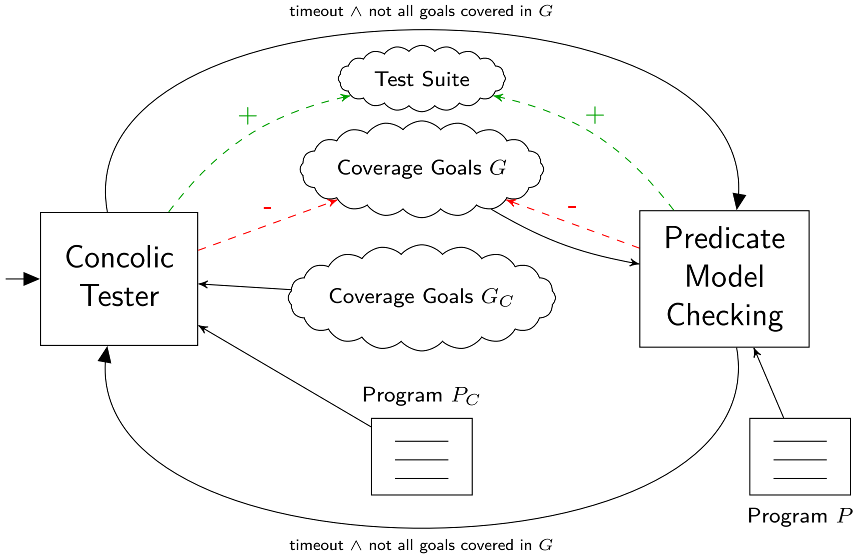
Abstraction-driven Concolic Testing
[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]



Abstraction-driven Concolic Testing
[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]

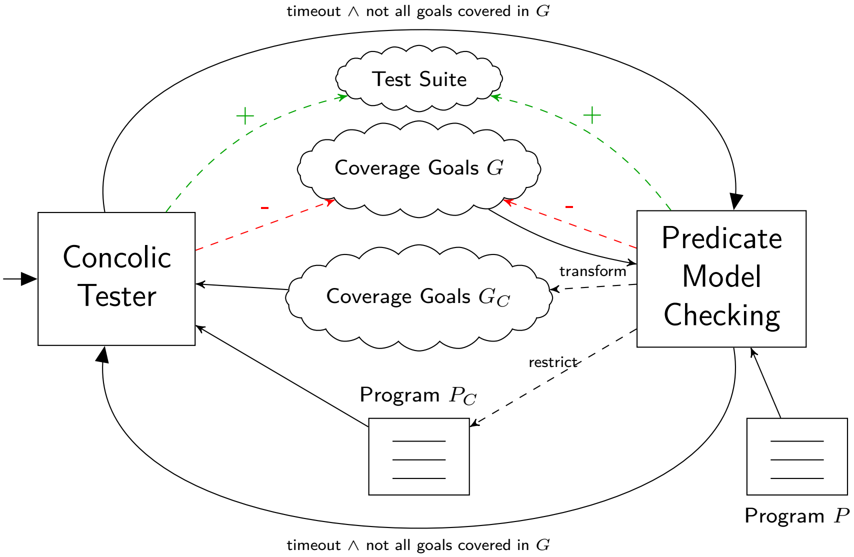


Abstraction-driven Concolic Testing
[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]



Abstraction-driven Concolic Testing

[P. Daca, A. Gupta, T. A. Henzinger, VMCAI2016]

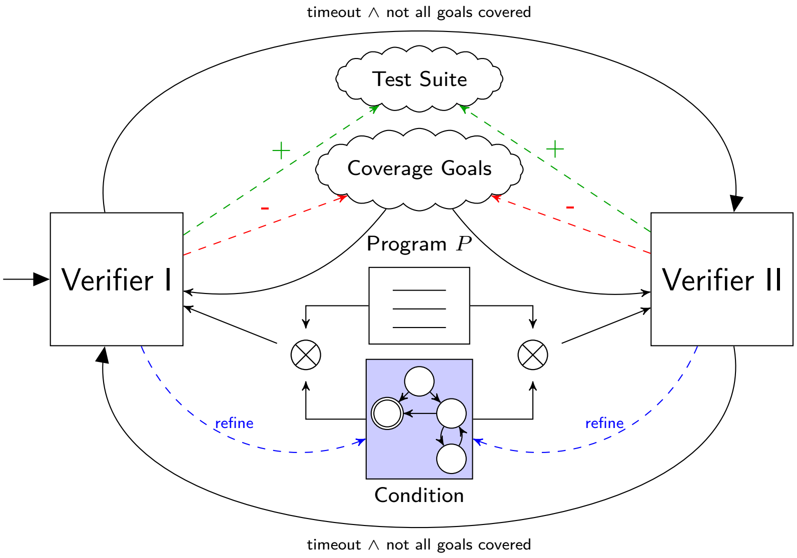


Idea: Generalized Abstraction-driven Testing

- Arbitrary verifiers, ≥ 2
- Cooperation (program restriction) via CMC
- Circular analyses

$$
\begin{array}{llcccl}
\text{Cooperation} & \{ & \text{none} & \text{unidirectional} & \text{bidirectional} & \} \\
 & & & \times & & \\
\text{Continuation} & \{ & \text{continue} & \text{restart} & \text{reuse precision} & \} \\
 & & & \times & & \\
\text{Timeout} & \{ & \text{uniform} & \multicolumn{2}{c}{\text{unbalanced}} & \}
\end{array}
$$

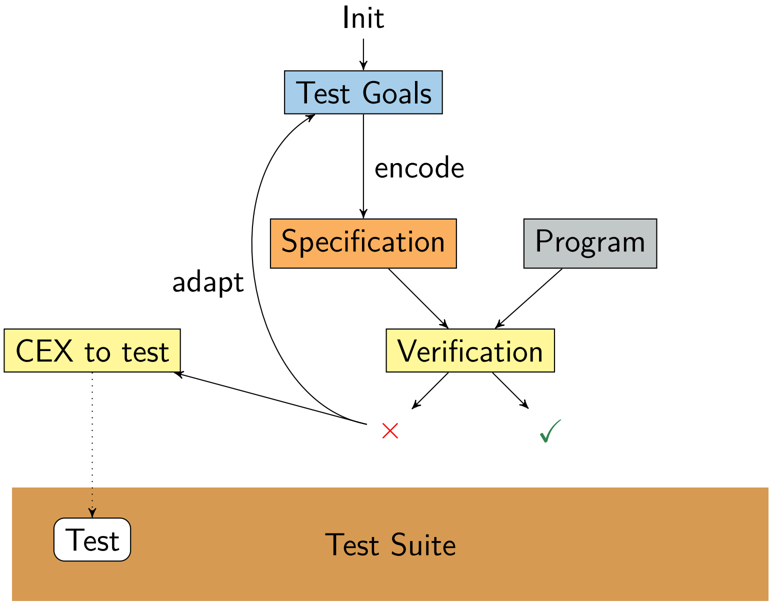
Abstraction-driven Testing – An Example



Implementing Abstraction-driven Testing

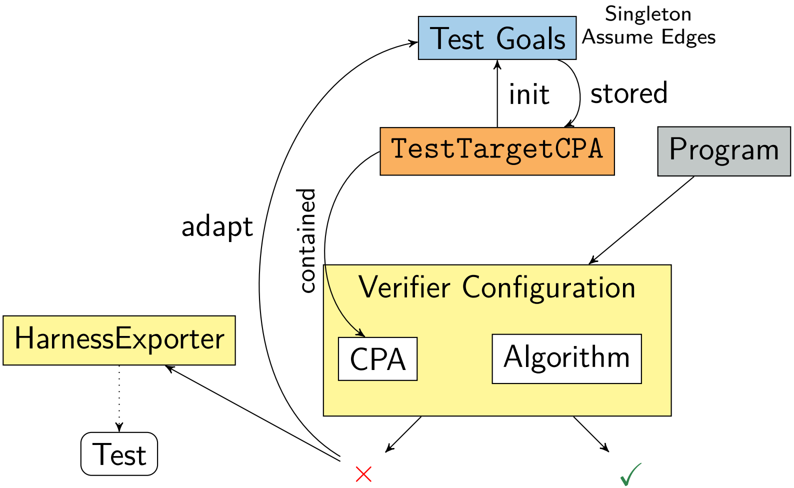
1. Test-case generation with verifiers
2. Circular analyses

Using Verification for Test-Case Generation



Test-Case Generation with CPAchecker

Realized by `TestCaseGeneratorAlgorithm`



TestTargetCPA

- Abstract domain

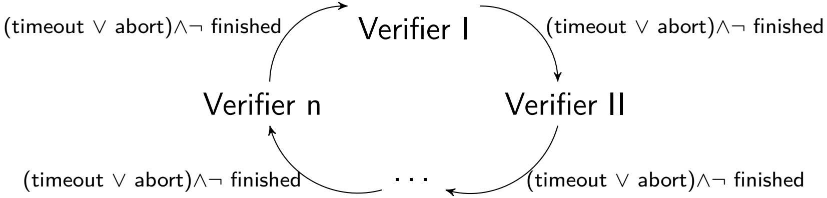
$$\begin{tikzcd}
 & \top \arrow[dl, dash] \arrow[dr, dash] & \\
\texttt{TARGET} \arrow[dr, dash] & & \texttt{NOTARGET} \arrow[dl, dash] \\
 & \bot &
\end{tikzcd}$$

- $\text{merge}^{\text{sep}}$ and stop^{sep}
- Precision adjustment: break if `TARGET` state

- Transfer relation

$$\begin{array}{ll}
(\cdot, g, \texttt{TARGET}) & \text{if } g \in TESTGOALS \\
(\cdot, g, \texttt{NOTARGET}) & \text{otherwise}
\end{array}$$

Circular Analyses: The `InterleavedAlgorithm`


Verifier I
(timeout ∨ abort)∧¬ finished
Verifier II
(timeout ∨ abort)∧¬ finished
· · ·
(timeout ∨ abort)∧¬ finished
Verifier n
(timeout ∨ abort)∧¬ finished

- Similar to `RestartAlgorithm`
- Continuation mode and time limit per component (::MODE_LIMIT)
- Rebuilds algorithm, but may reuse CPA or reached set

Research Questions

1. Configuration
   - Which timelimits?
   - Which cooperation form, continuation type?
2. Circular test generation better than single test generator?
3. Ciruclar test generation better than parallel?
4. Comparison with state-of-the-art
   - Better than exiting test generation tools?
   - Better than abstraction-driven concolic testing?

Evaluation Set-Up

Global limits: 15 min, 15 GB, 8 cores
All 6,703 SV-COMP programs except concurrency tasks

- Value and predicate analysis alone as baselines

- Combine value and predicate analysis
- Type (cooperation and continuation)

	restart	precision reuse	continue
none	✓	✓	✓
P→V	✓	✓	×
V→P	✓	✓	×
V↔P	✓	✓	×

- Timelimits: 10:10 50:50 100:100 250:250 20:80 80:20

Which Timelimits?

	restart	precision reuse	continue
none	250:250	20:80	20:80
P$\rightarrow$V	250:250	20:80	$\times$
V$\rightarrow$P	250:250	20:80	$\times$
V$\leftrightarrow$P	250:250	20:80	$\times$

Insights

- Avoid restart from scratch
- Prefer better single generator

How to Cooperate in General?

Which single test generator?

- Predicate

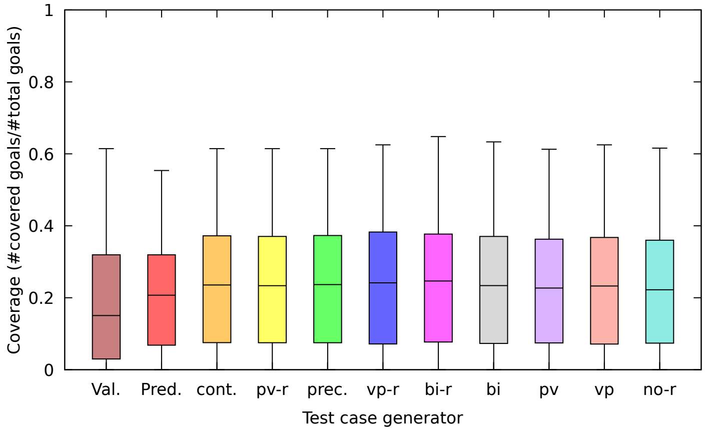
Which type of CMC cooperation?

- P$\rightarrow$V$\times$precision reuse

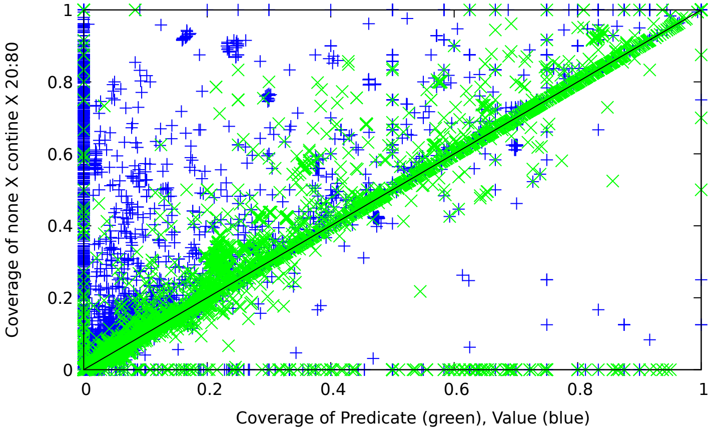
Which type of cooperation?

- none$\times$continue

Is Cooperation Better Than Single Generator?



Is Cooperation Better Than Single Generator?



Is Cooperation Better Than Single Generator?

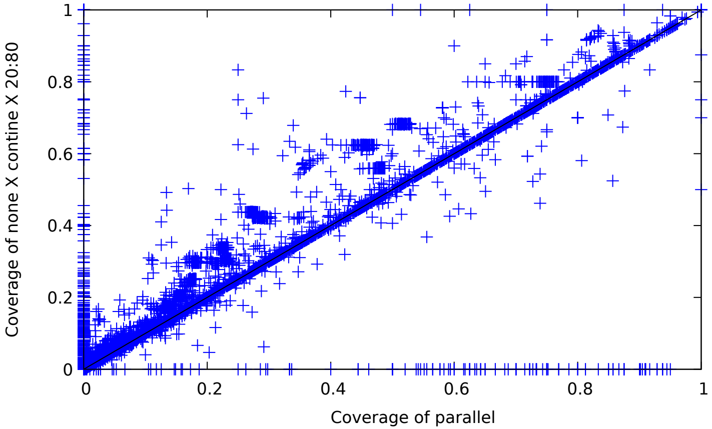
			Predicate	Value	Optimal Selector
250:250	restart	V→P	↓	↑	↓
		P→V	↘	↑	↓
		P↔V	↘	↑	↓
		none	↘	↑	↓
20:80	prec. reuse	V↔P	↘	↑	↓
		none	↗	↑	↓
		V→P	↗	↑	↓
		P→V	↑	↑	↓
	continue×none		↑	↑	↓
Optimal selector			↑	↑	↑

Is Cooperation Better Than Single Generator?

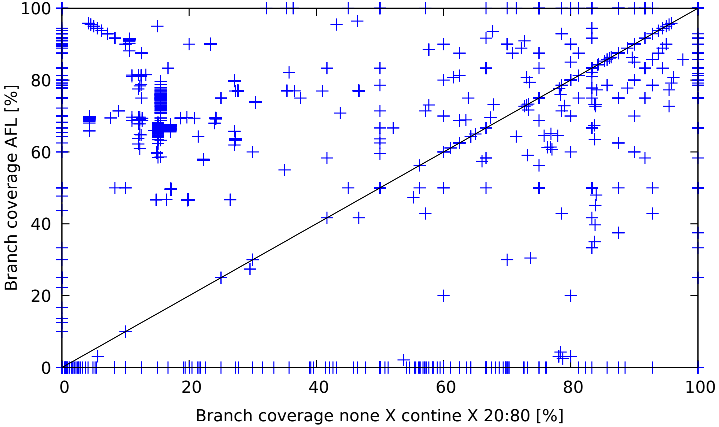
			Predicate	Value	Optimal Selector
250:250	restart	V→P	↓	↑	↓
		P→V	↘	↑	↓
		P↔V	↘	↑	↓
		none	↘	↑	↓
20:80	prec. reuse	V↔P	↘	↑	↓
		none	↗	↑	↓
		V→P	↗	↑	↓
		P→V	↑	↑	↓
	continue×none		↑	↑	↓
Optimal selector			↑	↑	↑

⇒ Reuse information and be cautious with program restriction

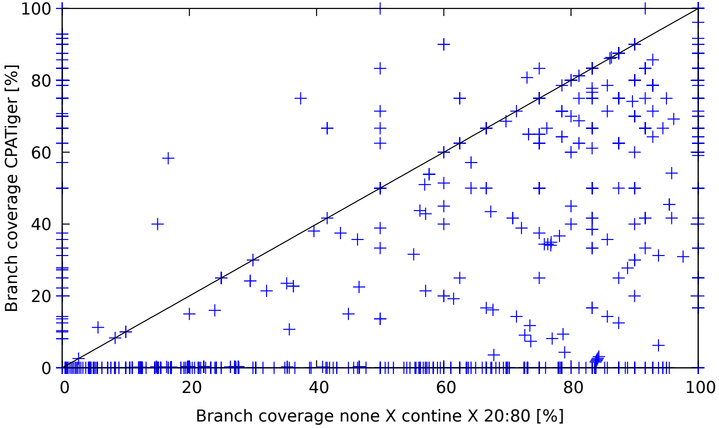
Is Cooperation Better than Running in Parallel?



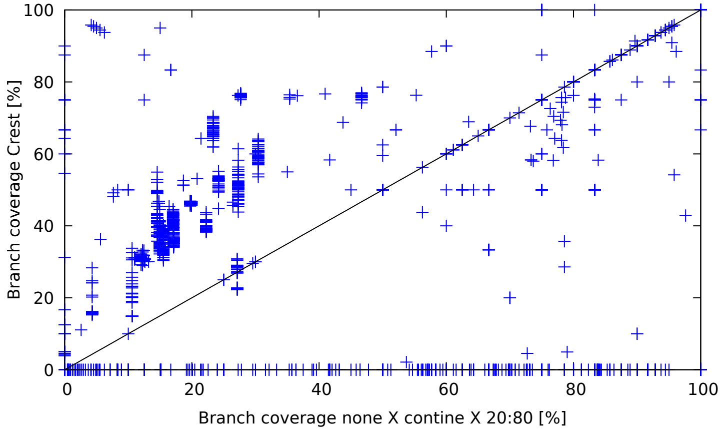
Comparison with Existing Test Tools
AFL



Comparison with Existing Test Tools
CPATiger

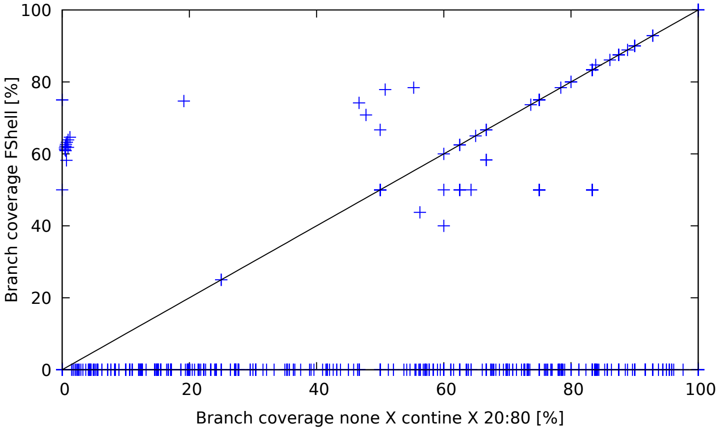


Comparison with Existing Test Tools
Crest

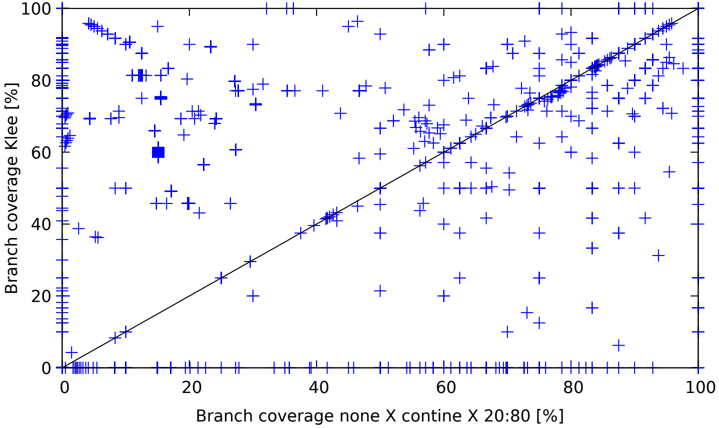


Comparison with Existing Test Tools

FShell



Comparison with Existing Test Tools
Klee



Conclusion

Cooperative test-case generation

- Cooperation can improve test-case generation result
- Sometimes better than existing test-case generators
- Realization
    - TestTargetCPA
    - TestCaseGeneratorAlgorithm
    - InterleavedAlgorithm

Open question

- Better than abstraction-driven concolic testing?