Korn

A C verifier based on Horn-clauses

https://github.com/gernst/korn

SV-COMP 2023, April 24

Gidon Ernst
gidon.ernst@lmu.de

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Korn - Background

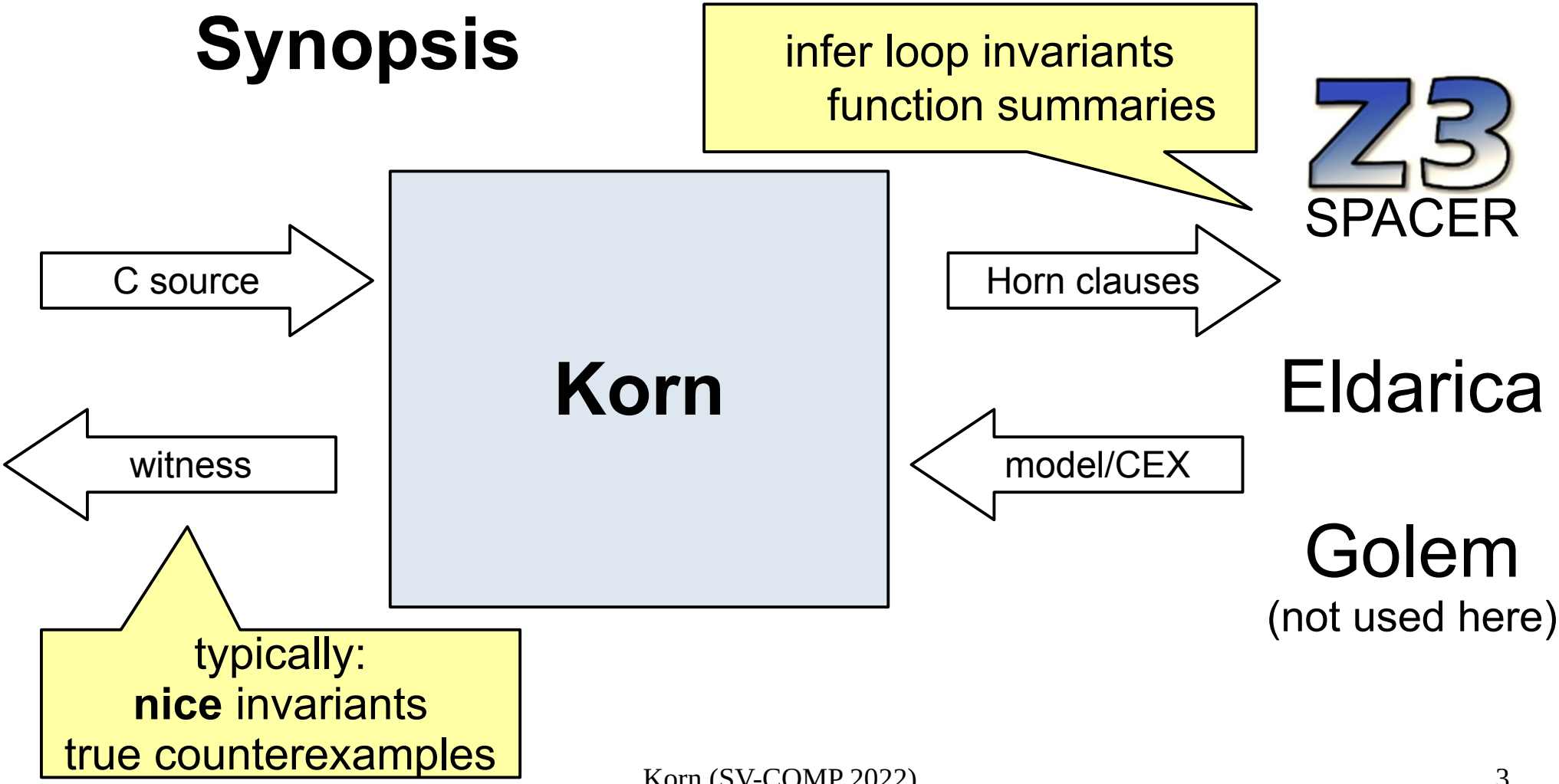
- Goal: investigate different loop encodings (notably: loop contracts, [VMCAI 2022])

  ⊕ easy to hack (Scala) ⊖ many C features missing

- SV-COMP: minor polishing over last year

- Participation: Recursive, Loops, ControlFlow, XCSP

Synopsis


infer loop invariants
function summaries
Z3
SPACER
C source
Korn
Horn clauses
Eldarica
witness
model/CEX
Golem
(not used here)
typically:
nice invariants
true counterexamples

Cheap Random Fuzzing

compile and run for the fun

- Many sv-benchmarks falsify with `__VERIFIER_nondet_*()` **small**
- Heuristic: uniform choice between a value in
  0 [0,1] [0,31] [0,1023]

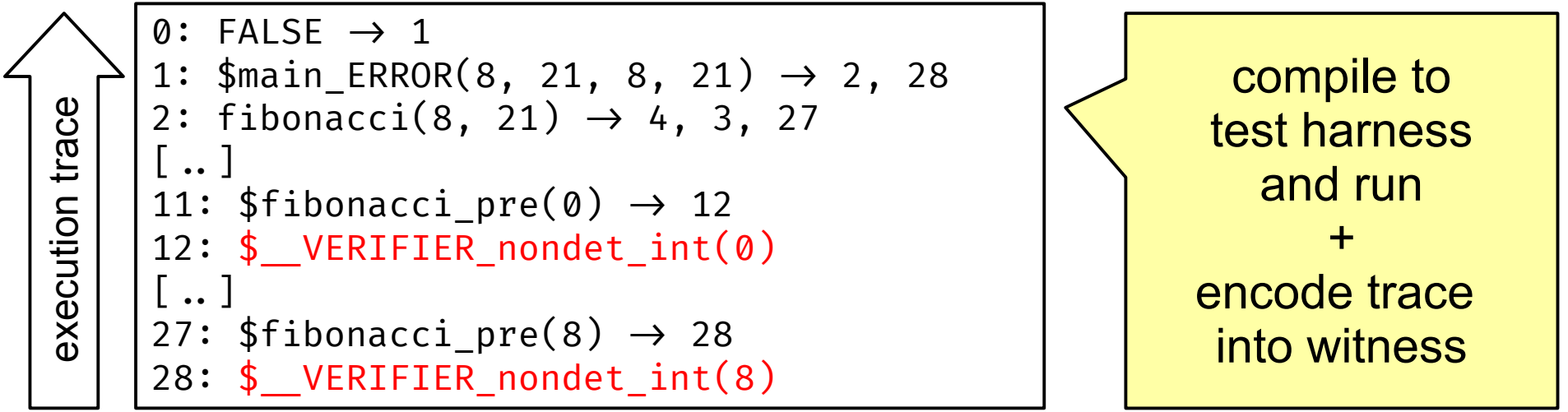
⊕ 210 problems solved in ~2s each

⊕ Avoids 1 unsound verdict (unsigned overflow)

Counterexample Validation

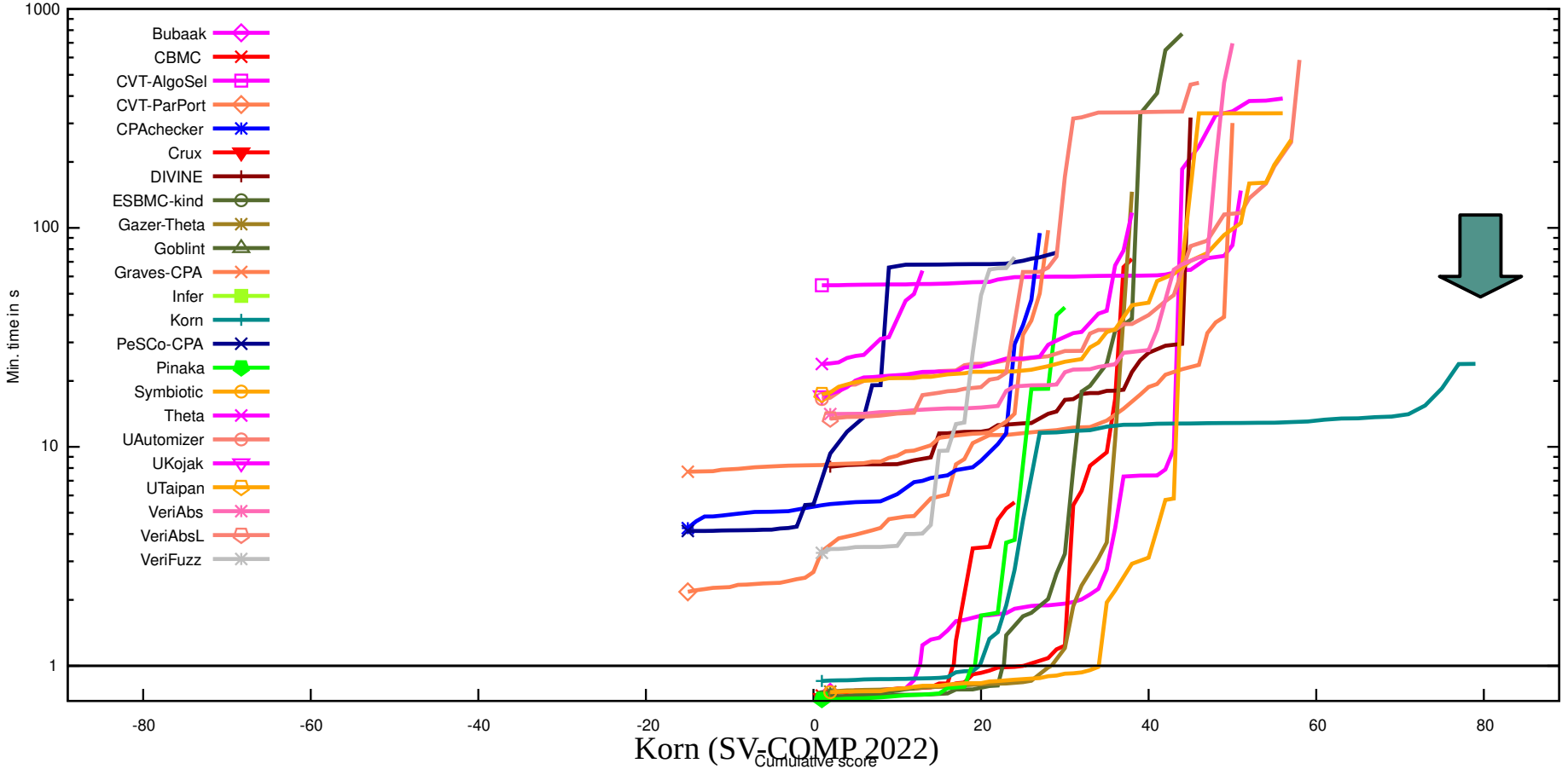
don't trust encoding and solvers

- Horn-clauses track `__VERIFIER_nondet_*()`



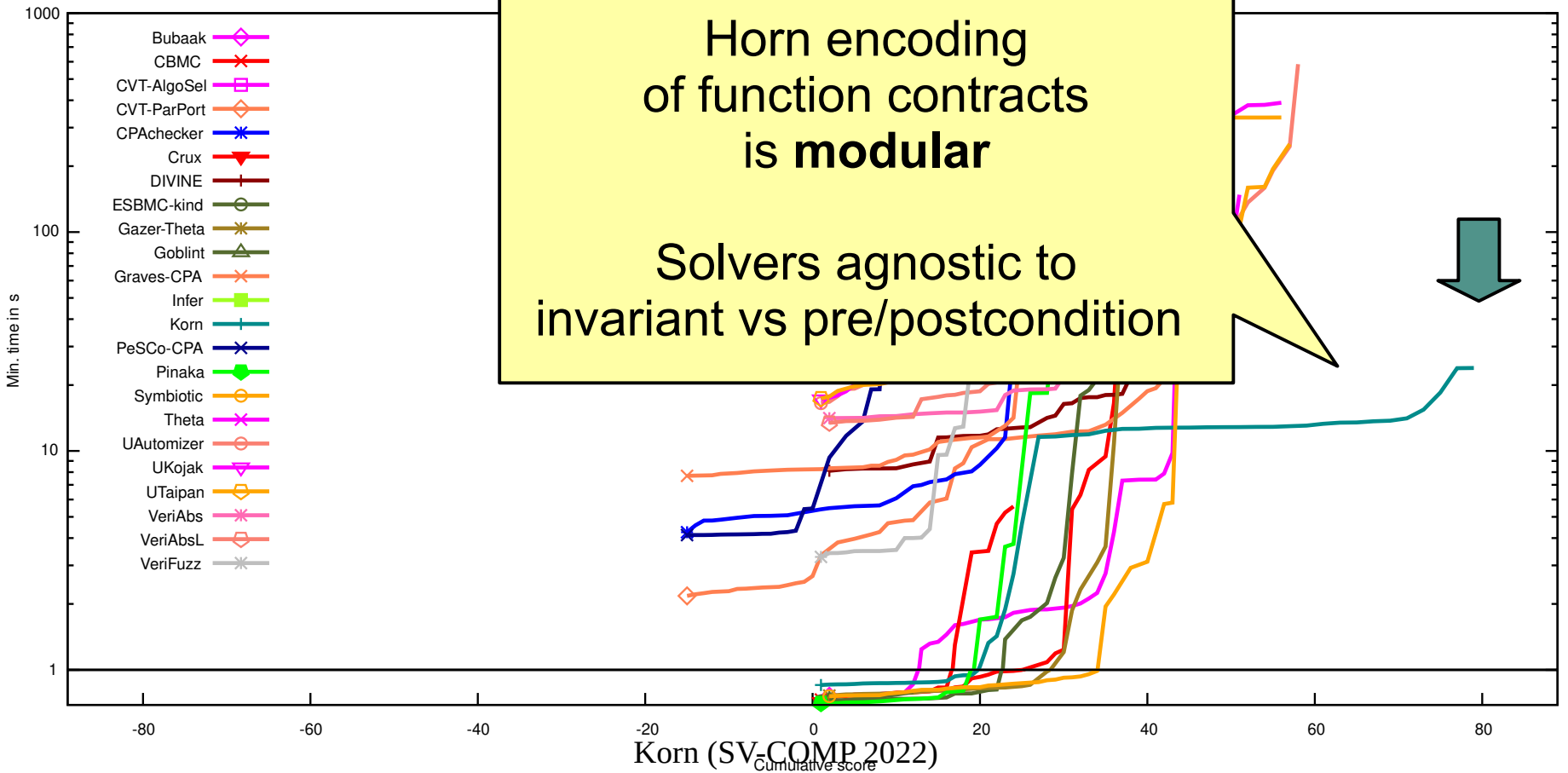
⊕ avoids a handful of incorrect false verdicts

ReachSafety-Recursive (#1)

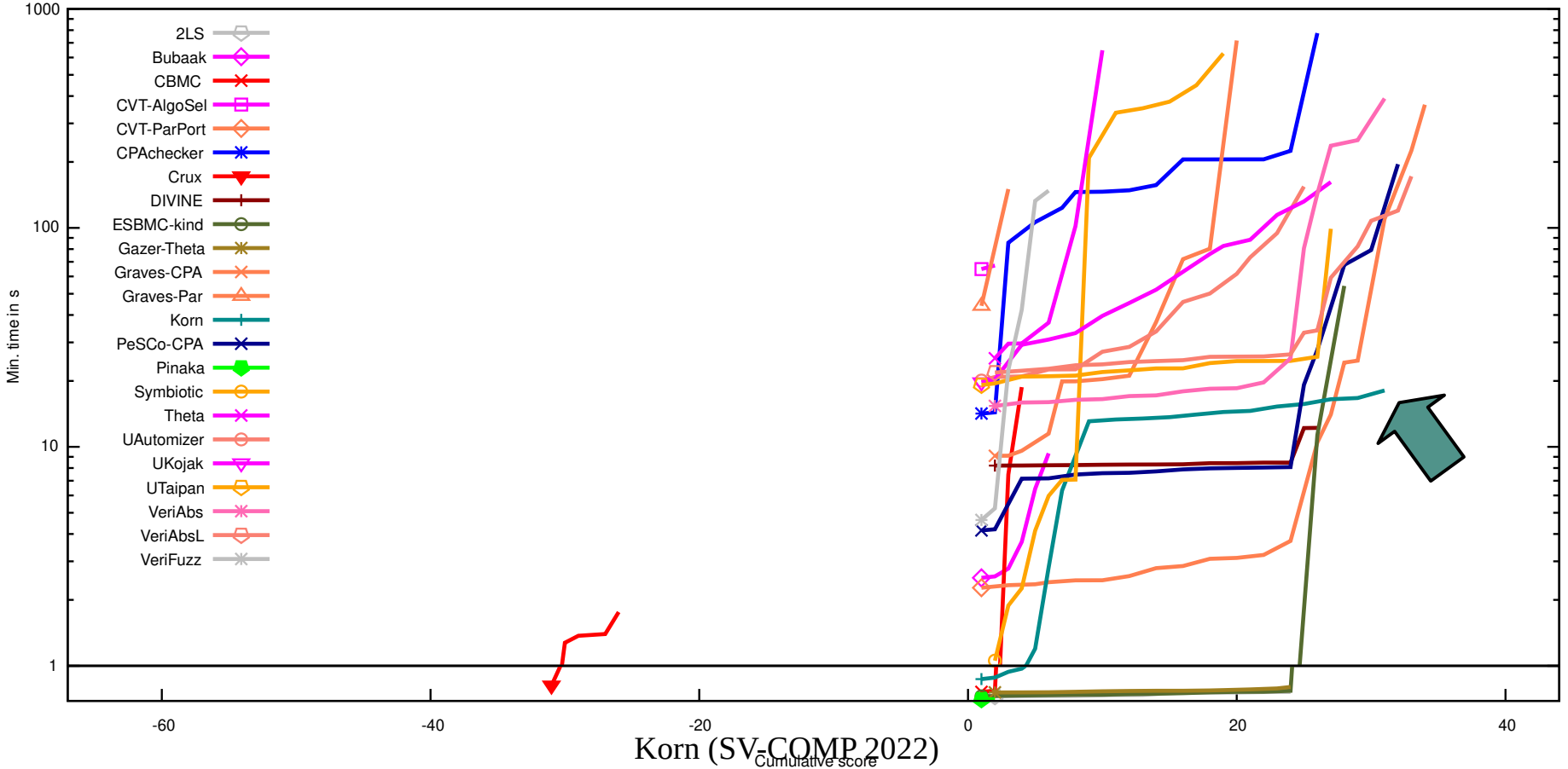


Korn (SV-COMP 2022)

ReachSafety-Recursive (#1)



ReachSafety-ControlFlow (tied #4)



ReachSafety-Loops and -XCSP

- Loops: Eldarica did not run (?)
  - Z3 solved 80 tasks, Eldarica can solve +208 tasks (hypothetical score: 755)

- XCSP: violations found by Z3 not be validated
  - lack of CEX (anyone knows how to get it?)
  - missing out on 50 violations (= best competitor)

Take-Away

https://github.com/gernst/korn

- Category Recursive:

  blind spot of others + right technique = success

- Horn solvers effective for numeric benchmarks
- portfolio pays off, including random sampling
- carefully look at pre-run results ;)

Horn-clause based Verification

(well-known, e.g. [Bjørner, Gurfinkel, McMillan, Rybalchenko 2015])

```
assume(i ≤ 0);
int i = 0;

while(i < n) {
  i++;
}

assert(i == n);
```

$$\exists\ inv.$$

second order

$$0 \leqslant n \wedge i=0 \implies inv(i,n)$$

$$i < n \wedge inv(i,n) \implies inv(i+1,n)$$

$$\neg(i < n) \wedge inv(i,n) \implies i = n$$