# Korn

## A C verifier based on Horn-clauses

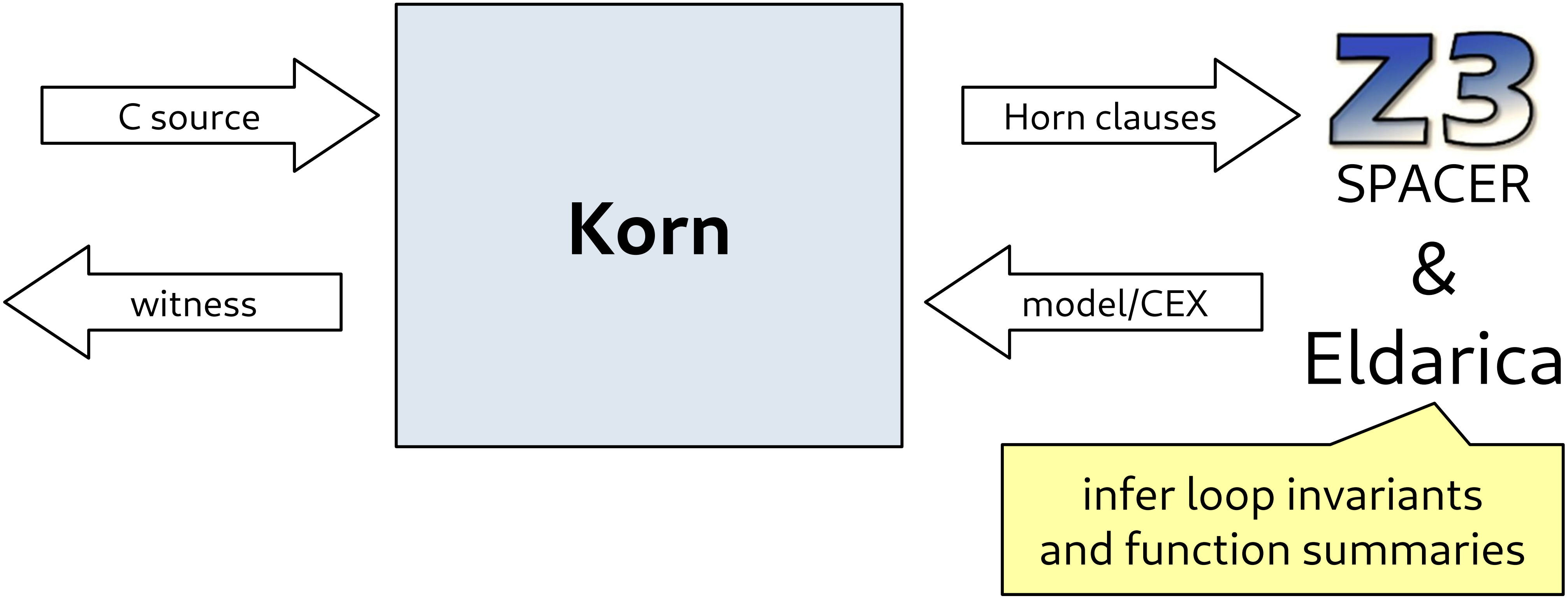
https://github.com/gernst/korn

**Gidon Ernst**

gidon.ernst@lmu.de

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

# Synopsis


C source
witness
Korn
Horn clauses
model/CEX
Z3
SPACER
&
Eldarica
infer loop invariants
and function summaries

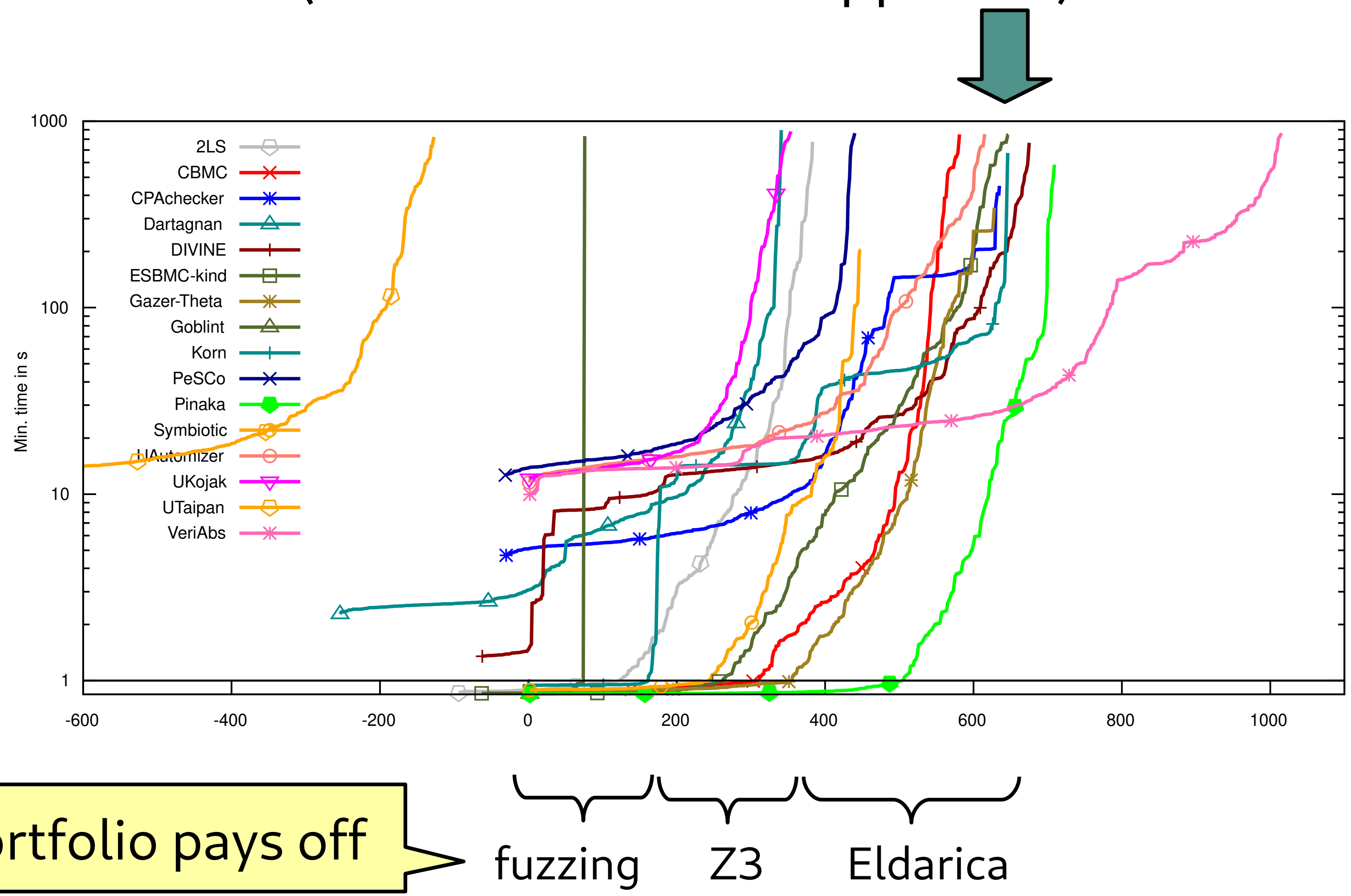
# Korn - Background

- Goal: investigate different loop encodings (contracts & invariants, arxiv.org/abs/2010.05812)
- SV-COMP (4 categories)
  - validate counterexamples (encoding limitations)
  - cheap random fuzzing (surprisingly effective)

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

# ReachSafety-Loops (close #5)

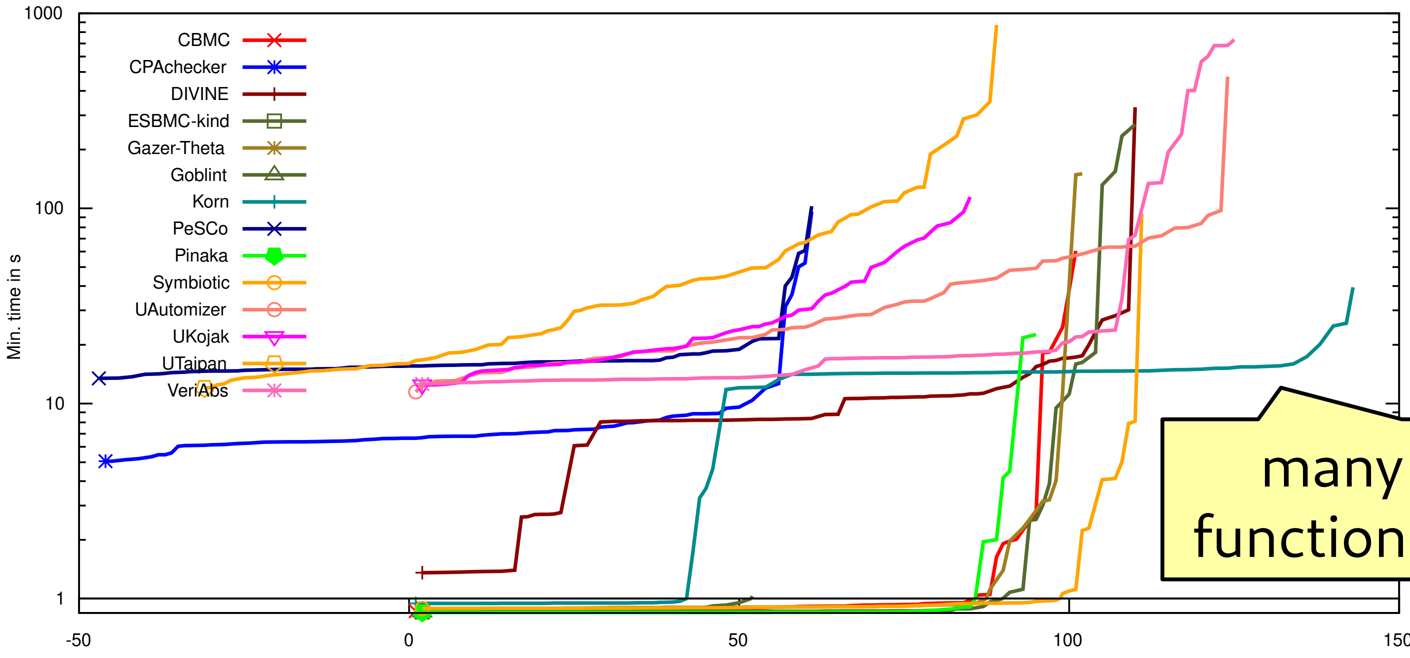
(646 of 768 tasks supported)




portfolio pays off

# ReachSafety-Recursive (#1)

(99 of 106 tasks supported)




many “simple” function summaries

many obvious bugs

fuzzing

Z3

# 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]

⊕ > 100 problems solved in < 10s

⊕ Avoids 1 unsound verdict (unsigned overflow)

# Counterexample Validation

don't trust encoding and solvers

- Horn-clauses track `__VERIFIER_nondet_*()`

execution trace

```
0: FALSE → 1
1: $main_ERROR(8, 21, 8, 21) → 2, 28
2: fibonacci(8, 21) → 4, 3, 27
[..]
11: $fibonacci_pre(0) → 12
12: $__VERIFIER_nondet_int(0)
[..]
27: $fibonacci_pre(8) → 28
28: $__VERIFIER_nondet_int(8)
```

compile to
test harness
and run
+
encode trace
into witness

⊕ avoids a handful of incorrect false verdicts

# Summary

https://github.com/gernst/korn

- Korn: experiment with Horn-clause encodings
- Solvers effective for arithmetic, bad with arrays

- Future: more of C, notably the heap
  evaluate polynomial abstract domain
  exploit loop structure (e.g. shrinking)

# 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);
```

second order

Horn: monotone in inv

$$\exists\ inv.$$

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

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

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