Checking Safety by Inductive Generalization of Counterexamples to Induction

Aaron R. Bradley and Zohar Manna
Stanford University

(Aaron is visiting EPFL and will be at CU Boulder)

Benchmark: `intel_005`
Solved: `vis-grab` (12 minutes, 178MB)

Our time: **11 seconds** (1 process)
Our memory: **13MB**

```
#latch vars: 170
#coi vars: 69
[1 1 0 0 0% 0% 0% 0% 0]
(1332 | !1662)
[2 1 0 1 9% 50% 49% 23% 25]
(1348 | !1668)
[3 1 0 2 23% 50% 33% 19% 71]
(!1342 | !1668)
[4 1 0 3 25% 42% 42% 18% 86]
(1624 | 1658 | !1626 | !1530 | !1666 | !1668)
[5 1 0 4 28% 60% 39% 14% 181]

...

[133 1 10 122 52% 58% 45% 1% 9000]
(1464 | 1586 | !1664 | !1668)
[134 1 10 123 52% 58% 45% 1% 9060]
(1574 | 1586 | 1638 | !1576 | !1372 | !1668)
[135 1 10 124 52% 58% 45% 1% 9143]
(1638 | !1662 | !1372 | !1668)
[136 1 10 125 52% 58% 46% 1% 9197]
Proved
Time: 11 (1)
VmPeak: 12820 kB
```

*(Source: **HWMCC'07**)*

Benchmark: `intel_006`
Solved: None

Our time: **5 minutes** (1 process)
Our memory: **92MB**

```
#latch vars: 350
#coi vars: 182
[1 1 0 0 0% 0% 0% 0% 0]
(l692 | !l1354 | !l1388)
[2 1 0 1 13% 22% 66% 18% 34]
(l922 | !l702 | !l738 | !l138
[3 1 0 2 30% 46% 46% 14% 88]
(l698 | !l926 | !l922 | !l1388)
[4 1 0 3 39% 46% 48% 12% 133]
(l764 | !l756 | !l894 | !l740 | !l1388)
[5 1 0 4 47% 43% 50% 10% 187]

...

[1144 1 60 1083 68% 49% 51% 1% 78386]
(l1384 | !l740 | !l1214 | !l768 | !l930 | !l1388)
[1145 1 60 1084 68% 49% 51% 1% 78453]
(l850 | l854 | !l1388)
[1146 1 60 1085 68% 49% 51% 1% 78515]
(l814 | l1014 | l1238 | !l886 | !l1388)
[1147 1 60 1086 68% 49% 51% 1% 78610]
Proved
Time: 285 (4)
VmPeak: 91748 kB
```

Benchmark: `intel_006`
Solved: None

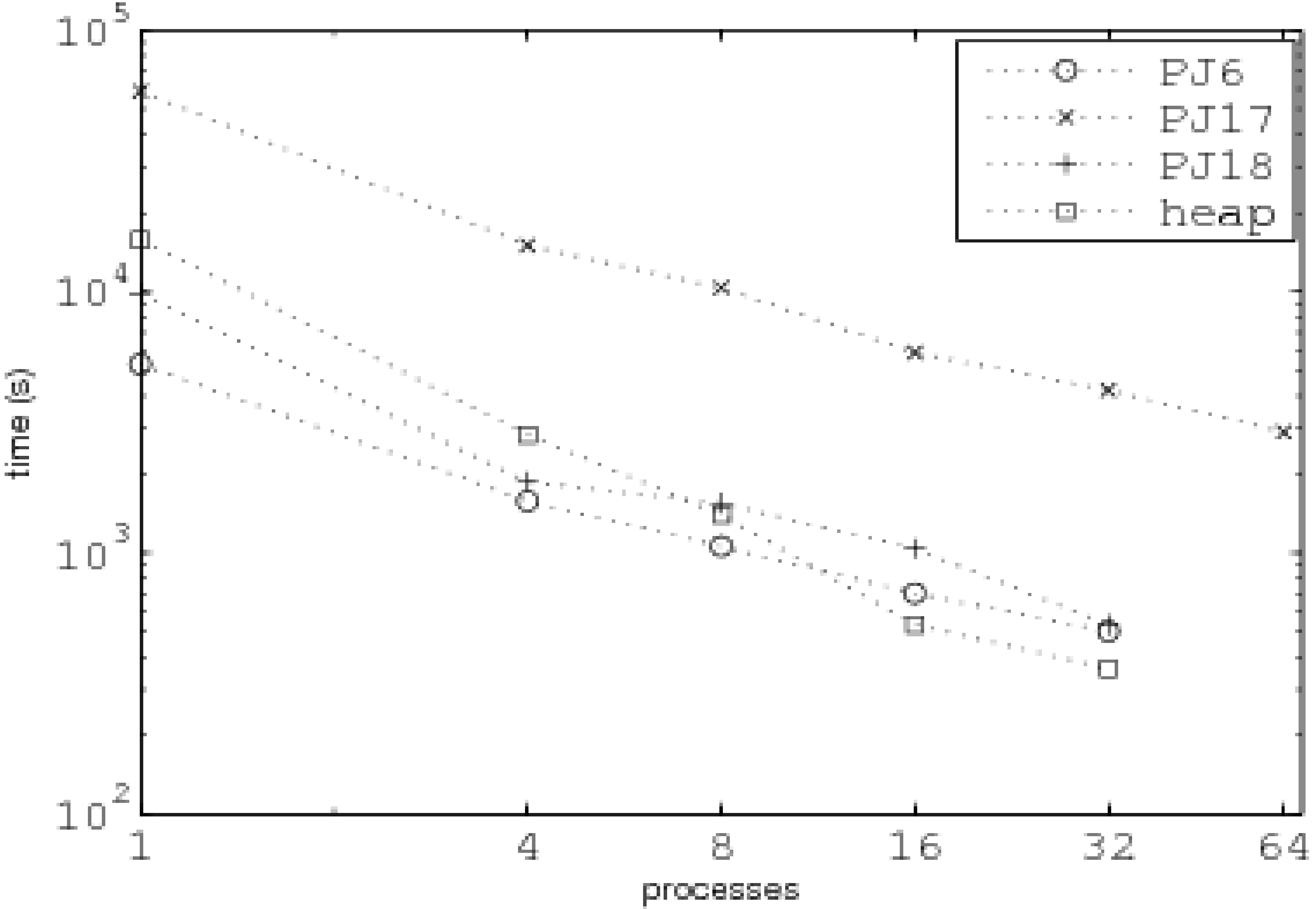
Our time: **1 minute** (8 processes)
Our memory: **30MB** (x 8)

```
ID: 979581
#latch vars: 350
#coi vars: 182
[1 1 0 0 0% 0% 0% 0% 0]
(l692 | !l1354 | !l1388)
[2 1 0 1 14% 22% 66% 17% 34]
(l706 | !l702 | !l1388)
[3 1 0 5 22% 29% 64% 14% 68]
(l810 | l874 | !l882 | !l1388)
[4 1 0 18 33% 40% 62% 11% 136]
(l780 | l1102 | l1166 | !l772 | !l1066 | !l1150 | !l1388)
[5 1 0 32 43% 45% 58% 8% 233]

...

[175 2 93 1167 66% 49% 50% 2% 12166]
(l800 | l806 | !l1056 | !l1388)
[176 1 94 1176 66% 49% 51% 2% 12249]
(l1086 | l1090 | !l1388)
[177 1 97 1177 66% 49% 51% 2% 12315]
[178 2 98 1178 66% 49% 50% 2% 12358]
Proved
Time: 49 (2)
VmPeak: 29204 kB
```

Parallel Scaling



```
ID: 962250
#latch vars: 1307
#coi vars: 608
[1 1 0 0 0% 0% 0% 0% 0]
(l2606 | !l5154 | !l5216)
[2 1 0 3 24% 21% 69% 11% 34]
(!l2616 | !l2612 | !l5216)
[3 1 0 5 31% 27% 61% 10% 57]
(l4430 | !l2616 | !l5216)
[4 1 0 14 42% 33% 55% 8% 100]
(l2616 | !l2634 | !l5216)
[5 1 0 18 45% 35% 54% 7% 122]

...

[238 1 0 1813 82% 47% 52% 0% 14661]
(l4426 | l4806 | !l3680 | !l5216)
[239 1 0 1821 82% 47% 52% 0% 14732]
(l3554 | l5018 | l5046 | !l5216)
[240 1 0 1828 82% 47% 52% 0% 14800]
(!l5114 | !l5110 | !l5216)
[241 1 0 1834 82% 47% 52% 0% 14856]
Proved
Time: 439 (4)
VmPeak: 37752 kB
```

Benchmark: `intel_007`
Solved: None

Our time: **8 minutes** (8 processes)
Our memory: **40MB** (x 8)

Other hard instances from HWMCC'07

`spec10-and-env (AMBA)`
8 processes: 1.5 hours, 900MB/process

`nusmv.reactor^2.C (TIP)`
1 process: 26 minutes, 22MB
8 processes: 4 **minutes**, 19MB/process

`nusmv.reactor^6.C (TIP)`
1 process: 43 minutes, 30MB
8 processes: **5 minutes**, 19MB/process

Not a “magic bullet”: utterly fails on
`cmu.dme[1/2].B`, `eijk.bs*`, ...
But perhaps a promising approach?

Different set of benchmarks in paper (PicoJava II).

The Verification Team Analogy

Goal: Inductive strengthening of property

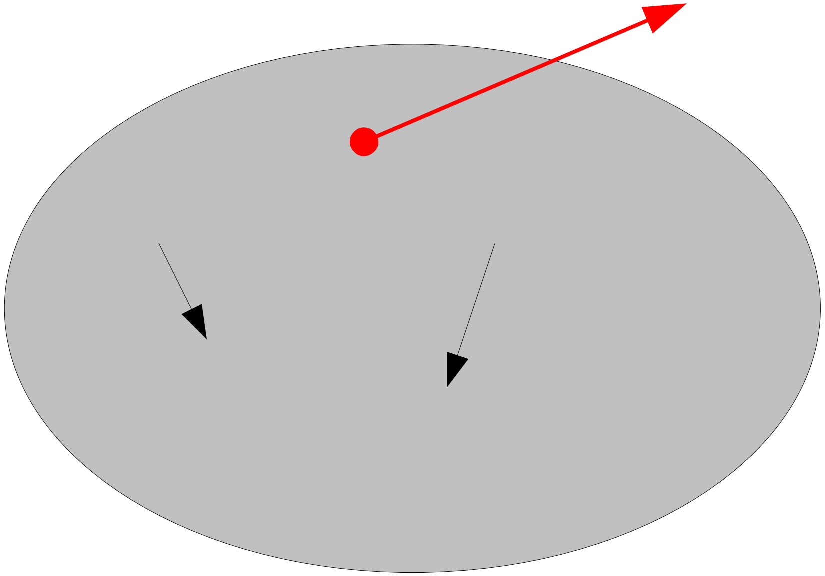
Verification Team	**Inductive Generalization**
1. Individuals	1. Processes
2. Lemmas	2. Inductive Clauses
3. Property	3. Property

Lemma: Summary of **observation** and **proof**

Lemma: Inductive Clause

1. Counterexample to induction:

No counterexample?
Then property is valid.



```
State s:   !12606 & ... & 15154 & ... & 15216
Clause ~s: 12606 | ... | !15154 | ... | !15216
```

Lemma: Inductive Clause

2. Minimal inductive subclause:

Original Clause ~s:

```
12606 | ... | !15154 | ... | !15216
```

608 literals. Inductive? Maybe, maybe not.

Minimal Inductive Subclause:

```
12606 | !15154 | !15216
```

3 literals (informative!).
Inductive relative to property and previous clauses.

Inductive Generalization

Clause ~s: `12606 | ... | !15154 | ... | !15216`

Maximal inductive subclause:

- Unique.
- Best approximation of computing preimage to fixpoint.
- Weak: Excludes "only" states that can reach s.

Minimal inductive subclause:

- Not unique.
- Minimal: Strict subclauses are not inductive.
- Strong: Also excludes many states that cannot reach s.

Inductive explanation of why s and similar states are unreachable.

Discovery of MI Subclause

```
[1 1 0 0 0% 0% 0% 0% 0]
(l2606 | !l5154 | !l5216)
[2 1 0 3 24% 21% 69% 11% 34]
(!l2616 | !l2612 | !l5216)
[3 1 0 5 31% 27% 61% 10% 57]
(l4430 | !l2616 | !l5216)
[4 1 0 14 42% 33% 55% 8% 100]
(l2616 | !l2634 | !l5216)
[5 1 0 18 45% 35% 54% 7% 122]
```

608 literals.
But <100 SAT problems/iteration.

Discovery of MI Subclause

Many “easy” SAT queries.

1. O(n) SAT queries to find maximal IS c_1.
   In practice: many fewer than n

2. O(m lg n) SAT queries to find “small” m-literal inductive subclause c_2 of c_1.
   In practice: m is very small

3. Brute force to guarantee minimality.
   In practice: Algorithm 2 minimizes effects

Related Work

- Interpolation-based model checking [McMillan]
- CEGAR (Jain et al., Clarke et al., ...)
  Abstract transition relation

- BMC, k-induction [Biere et al., Sheeran et al., ...]
  Reduce to large SAT/QBF queries.

- Strengthening in k-induction
  [deMoura et al., Vimjam et al., Awedh et al., ...]
  Based on preimage of counterexample.
  Weak, so k-induction is main principle.

Ongoing & Future Work

1. Combine with k-induction for **small** k.
   Better counterexamples to induction.
   Stronger clauses.
   Balance k and ease of SAT queries.

2. Combine with BMC for better debugging.
   Add clauses to BMC SAT query online.

3. Other types of lemmas?

4. Better engineering.
   Obstacle to handling large Intel benchmarks.

Conclusions

- Principle: Iterative discovery of lemmas.
  Control resource usage.
  Run in parallel.

- Principle: Use induction to generalize.

- Mechanism:
  Fast discovery of minimal inductive subclauses.

Questions? Comments?