Augmenting Predicate Analysis With Auxiliary Invariants

Thomas Stieglmaier

September 23, 2016

University of Passau

Motivation

Predicate Analysis

- SMT-based
- Abstraction of program, computed from a set of predicates π
- CEGAR for refining π
- Craig interpolation for discovering precision increments

Motivation

Predicate Analysis

- SMT-based
- Abstraction of program, computed from a set of predicates π
- CEGAR for refining π
- Craig interpolation for discovering precision increments

- precision π
- path formula ϕ
- abstraction formula ψ

Abstraction computation

- $\psi' = (\phi \wedge \psi)^{\pi}$
- $\phi = \text{TRUE}$

Motivation — Invariants

Generating Invariants

- Several tools available: InvGen, Daikon
- Often not SMT-based

Motivation — Invariants

Generating Invariants

- Several tools available: InvGen, Daikon
- Often not SMT-based

Use invariants in other analyses

- Add new (helpful) information to a predicate analysis
- Speed up the analysis
  - less refinements
  - less dependent on interpolants

Predicate Analysis — Adding Invariants

$$\psi' = (\phi \wedge \psi \wedge \textbf{INV})^{\pi}$$

Predicate Analysis — Adding Invariants

$$\psi' = (\phi \wedge \psi)^{\pi \cup \{\textbf{INV}\}}$$

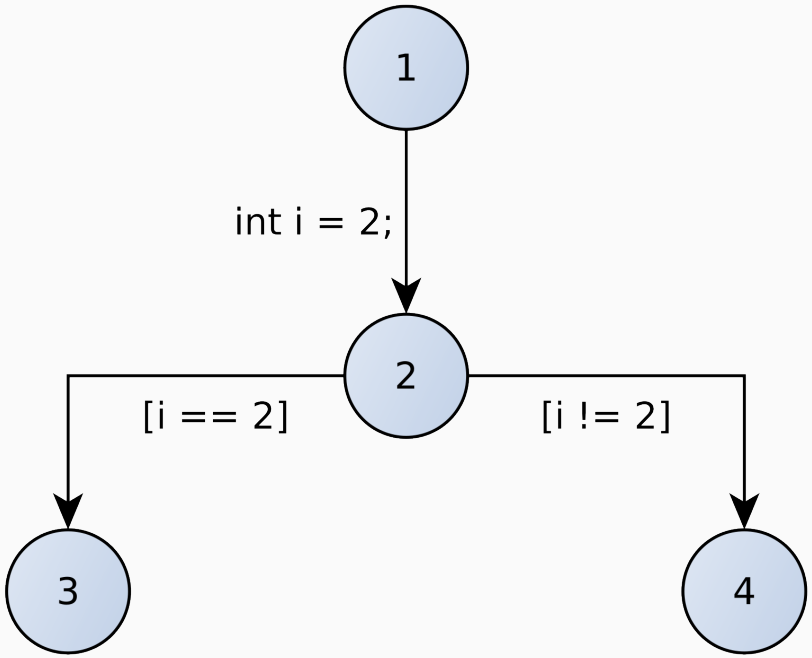
Predicate Analysis — Adding Invariants

$$\psi' = (\phi \wedge \psi)^{\pi} \wedge \textbf{INV}$$

Predicate Analysis — Adding Invariants

$$\psi' = (\phi \wedge \psi \wedge \textbf{INV})^{\pi \cup \{\textbf{INV}\}} \wedge \textbf{INV}$$

Predicate Analysis — Example



Location ②

- Abstraction location
- $\pi = \{i < 10\}$
- invariant $i = 2$

Predicate Analysis — Example

Strategy	New Abstract State	Possible Transitions
No Inv	$(i < 10, \quad \text{TRUE})$	$2 \to 3, 2 \to 4$
Prec	$(i = 2 \wedge i < 10, \quad \text{TRUE})$	$2 \to 3$
PF	$(i < 10, \quad i = 2)$	$2 \to 3$
AF	$(i = 2 \wedge i < 10, \quad \text{TRUE})$	$2 \to 3$
Prec + PF	$(i = 2 \wedge i < 10, \quad i = 2)$	$2 \to 3$
Prec + AF	$(i = 2 \wedge i = 2 \wedge i < 10, \quad \text{TRUE})$	$2 \to 3$
PF + AF	$(i = 2 \wedge i < 10, \quad i = 2)$	$2 \to 3$
Prec + PF + AF	$(i = 2 \wedge i = 2 \wedge i < 10, \quad i = 2)$	$2 \to 3$

Auxiliary Invariants

- fast computation
- high success rate
- useful invariants

Auxiliary Invariants

- fast computation
- high success rate
- useful invariants

$\rightarrow$ no negative impact on the main analysis

Auxiliary Invariants — Lightweight Heuristics

PredicateCPA specific

- Inductive weakening of path formulas
- Checking conjuncts of path formulas on invariance
- Checking interpolants on invariance

Auxiliary Invariants — Lightweight Heuristics

PredicateCPA specific

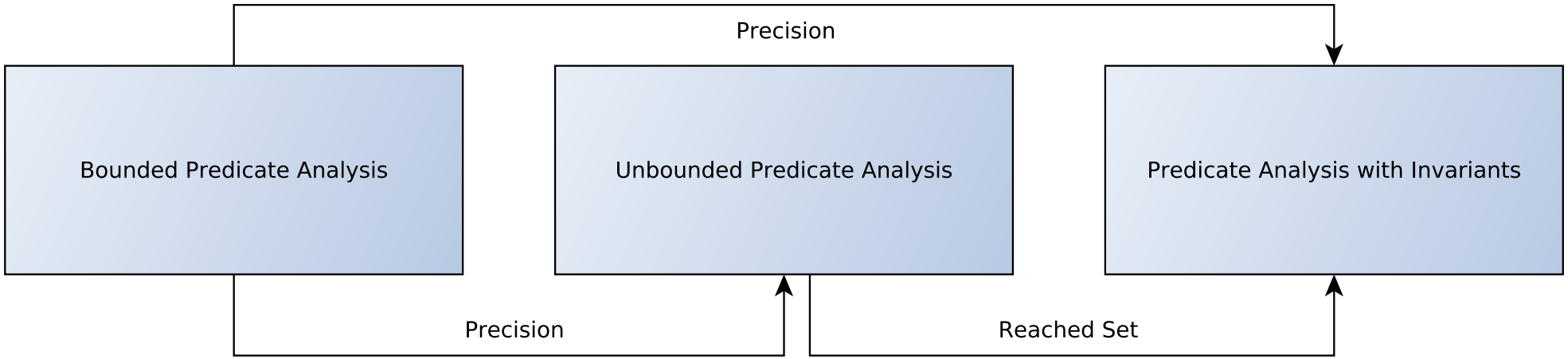
- Inductive weakening of path formulas
- Checking conjuncts of path formulas on invariance
- Checking interpolants on invariance

Applicable to other analyses

- Path invariants

Auxiliary Invariants — Sequential Analyses

Compute invariants from reached sets of earlier analyses



Auxiliary Invariants — Parallel Analyses

- k-induction uses concurrently running invariant generation
  - ↯ not usable for other concurrent analyses

→ new CPAchecker feature

- Algorithm for executing several analyses in parallel
- Communication between analyses via reached sets

Handling Invariants in the PredicateCPA

- One manager class
  - Exposes general methods for retrieving and generating invariants
  - Hides exact configuration
  - Lazy computation of invariants during refinement
  - Mixing generation and usage strategies possible

Handling Invariants in the PredicateCPA

- One manager class
  - Exposes general methods for retrieving and generating invariants
  - Hides exact configuration
  - Lazy computation of invariants during refinement
  - Mixing generation and usage strategies possible
- Two users
  - Refinement (precision increment)
  - PrecisionAdjustment (path -and abstraction formula)

Evaluation — Environment

- 2.6 GHz Octa Core CPUs (Intel E5-2650 v2)
- 8 GB memory
- 300 s or 600 s CPU time
- *trunk* r23084
- Measured with BenchExec

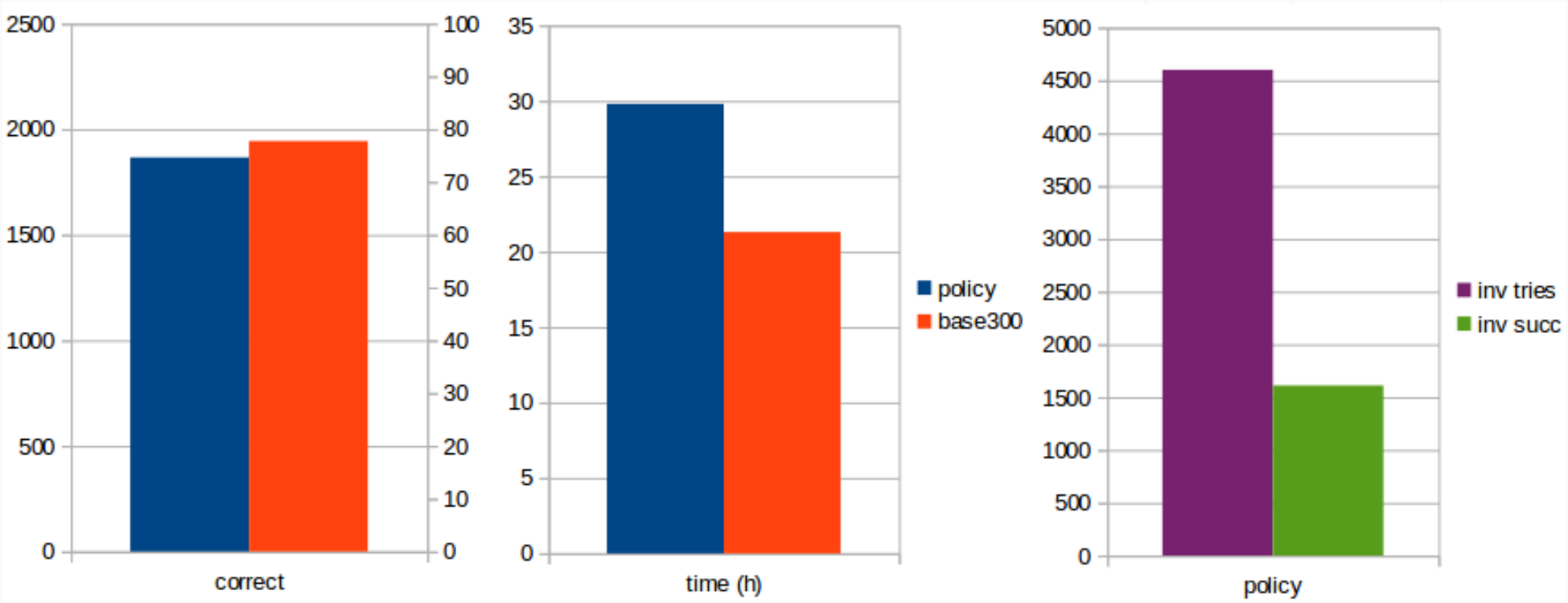
- 3488 verification tasks taken from SV-COMP'16

Evaluation — Heuristics

- Inductive weakening and checking conjuncts of path formulas failed
- Checking interpolants on invariance is very slow due to prefix generation
- Path invariants are too slow overall, but good on tasks in the loops category

Evaluation — Path Invariants

- Two configurations:
  - Predicate Analysis + Path Invariants with InvariantsCPA
  - Predicate Analysis + Path Invariants with PolicyCPA



Evaluation — Path Invariants

```
1  int main() {
2    int i;
3    for (i = 0; i < 1000000; i++) ;
4    assert(i == 1000000);
5    return 0;
6  }
```

↯ Interpolation unrolls the loop

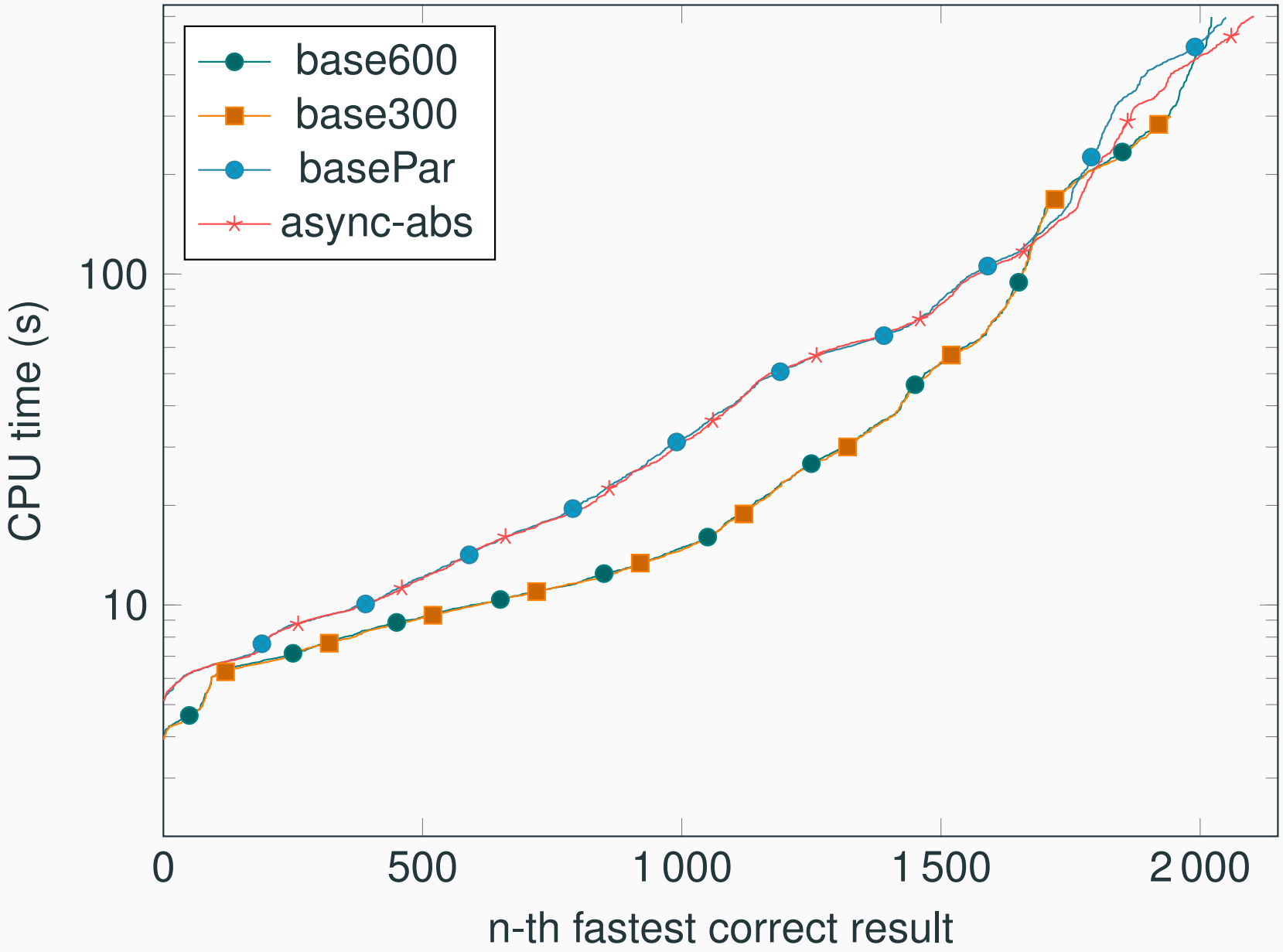
✓ found invariant: `i = 1000000` for location of `assert` call

Evaluation — Parallel Analyses

- Combination of:
  - An analysis with the PredicateCPA, and
  - An analysis with the InvariantsCPA (continuously-refined)
- 600 s CPU time (300 s per analysis)
- 7 configurations: **abs**, **prec**, **path**, **abs-path**, ...

- 3 baselines
  - 300 s and 600 s predicate analyses
    base300, **base600**
  - 600 s parallel analysis without invariant generation
    basePar

Evaluation — Parallel Analyses



Evaluation — Parallel Analyses

- all baselines are strictly worse than configurations with invariants
- **async-abs** is the best configuration
  - 4 % better than **base600**
  - 8 % better than **base300**
  - 3 % better than **basePar**

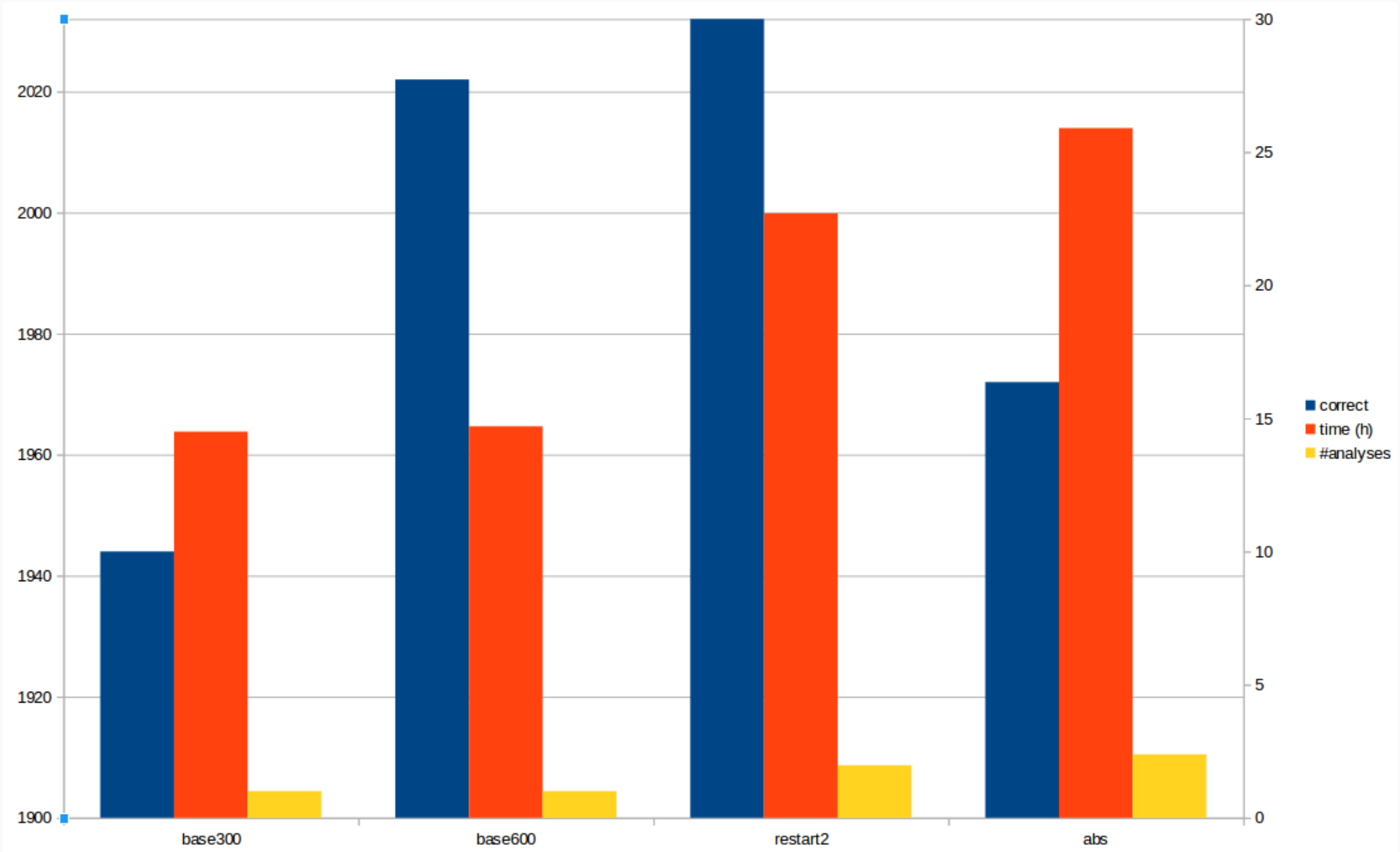
  $\rightarrow$ wall time is comparable to **base300**
- **async-prec** is slow
- **async-prec-path** almost as good as **async-abs**

Evaluation — Sequential Analyses

- Combination of:
  - bounded predicate analysis (100 s)
  - unbounded predicate analysis without refinement (100 s)
  - predicate analysis using invariants (300 s)
- 7 configurations (invariants): **abs**, **prec**, **path**, **abs-path**, ...
- 1 configuration (only precision): **restart2**

- 2 baselines, 300 s and 600 s predicate analyses
  base300, **base600**

Evaluation — Sequential Analyses



Conclusion & Outlook

- Heuristics for invariant generation need more time than expected
- More intelligent heuristics needed:
  - When should invariants be generated
  - Filtering of found invariants

Conclusion & Outlook

- Heuristics for invariant generation need more time than expected
- More intelligent heuristics needed:
  - When should invariants be generated
  - Filtering of found invariants

✓ Combination of analyses increases performance

✓ Performance is even better if the analyses communicate

→ Aim: Make communication easier usable

Path Invariants — Table (1)

Table 1: Details on analyses using path invariants for generating auxiliary invariants and their baseline

	correct		wrong	Invariants (equal)			CPU time (h)		
	safe	unsafe	safe	time (h)	tries	succ	all	correct	equal
base300	1 391	553	27				149	26.0	21.3
path-inv	1 327	519	27	2.36	4 719	1 428	162	31.0	30.5
path-policy	1 337	529	27	3.84	4 600	1 611	161	31.4	29.8
400s-inv	1 364	575	27				196	35.6	
400s-policy	1 371	576	27				196	34.7	

Path Invariants — Table (2)

Table 2: A selection of tasks and their results with path invariants

file name	path-inv	path-policy
loop-acceleration/array_true-unreach-call3.i	✓	✗
loop-acceleration/functions_true-unreach-call1.i	✗	✓
loop-acceleration/nested_true-unreach-call1.i	✓	✗
loop-acceleration/simple_true-unreach-call1.i	✗	✓
loop-new/count_by_1_true-unreach-call.i	✓	✗
loop-new/count_by_1_variant_true-unreach-call.i	✓	✗
loop-new/count_by_nondet_true-unreach-call.i	✗	✓

Parallel Analyses — Table

Table 3: Details on all parallel analyses using invariants and their baselines

	correct		wrong		Main Succ	Wall time (h)		CPU time (h)	
	true	false	true	false	correct	all	equal	all	equal
base300	1 391	553	0	27	1 944	128	13.8	149	20.9
base600	1 434	588	0	27	2 022	240	13.9	262	21.1
basePar	1 509	541	0	18	1 109	152	15.6	281	39.9
abs	1 532	572	0	18	1 154	147	14.4	276	38.2
path	1 536	561	1	17	1 148	146	14.2	274	37.9
prec	1 526	549	0	18	1 108	149	15.3	279	39.6
prec-path	1 525	561	1	17	1 111	148	15.1	278	39.4
abs-path	1 528	568	1	18	1 148	146	14.4	275	38.4
prec-abs	1 526	557	0	18	1 110	149	15.2	279	39.4
prec-abs-path	1 531	551	1	18	1 106	148	15.0	278	39.5

Sequential Analyses — Table

Table 4: Details on all sequential combinations of analyses using invariants and their baselines

	correct		wrong	∅ Analyses	Wall time (h)					
	true	false	false		Alg1	Alg2	Alg3	all	correct	equal
base300	1 391	553	27	1.00				128	17.9	14.5
base600	1 434	588	27	1.00				240	26.7	14.7
restart2	1 420	612	27	1.97	12.3	8.84		182	29.1	22.7
abs	1 415	557	27	2.38	12.3	3.35	8.73	201	32.3	25.9
path	1 416	547	28	2.38	12.3	3.39	8.65	200	30.9	25.9
prec	1 409	550	27	2.38	12.3	3.33	9.15	202	31.7	26.3
prec-path	1 409	557	28	2.38	12.3	3.40	8.89	201	31.7	26.1
abs-path	1 414	555	28	2.38	12.3	3.35	8.66	200	31.5	25.9
prec-abs	1 407	555	27	2.38	12.3	3.36	9.21	202	32.0	26.4
prec-abs-path	1 414	552	26	2.38	12.3	3.35	9.13	201	31.8	26.3