Evaluating Design Tradeoffs in Numeric Static Analysis for Java

Real leader of this effort!

Shiyi Wei, Piotr Mardziel, Andrew Ruef,
Jeffrey S. Foster, and Michael Hicks

Work done while all authors at

University of Texas at Dallas, Carnegie Mellon,

PLUM Lab for **P**rogramming **L**anguages Research
at the **U**niversity of **M**aryland

Numeric Static Analysis (for Java)

- Prove numeric properties about programs, to show
  - array indexes in bounds,
  - division never by zero,
  - running times do not exceed limit, etc.
    - Previous work: Use numeric analysis to compute running times, and see if they depend on a secret's value (PLDI'17)
  - Often framed as **abstract interpretation**
- Challenge: Analyzer that works for **full Java**
  - Aim to be sound(y), precise, scalable

The Problem

- No prior analysis/paper soundily models
  - the heap,
  - method calls, and
  - numeric domains
  - in a way that scales, and requires few/no annotations
- Many design choices in building such an analysis
  - Lots of folklore about these choices
    - Possibly overfit to evaluation target programs?
  - Most papers consider only single choices
    - Not their interactions

Prior Work

- PAGAI: numeric analysis, but no heap
- Fu (2014) considers both, but no method calls
- Ferrara; Magill; McCloskey et al; Chang and Rival: Sophisticated numeric+heap invariants but
  - not scalable, requires annotations, may be missing language features (e.g., method calls or loops)
- Little work analyzes interacting tradeoffs
  - ASTREE has many ideas, but not considered systematically (just final result indicated in paper)
  - Some one-offs; eg., polyhedra vs. intervals

Our work

- We built a Java analysis tool that has multiple implementations of five analysis components
  - numeric domain (2)
  - heap abstraction (3)
  - abstract object representation (3)
  - interprocedural analysis order (3)
  - level of context sensitivity (3)

  } 162 configurations

- Ran each on DaCapo, measured performance and precision on proving array indexes in bounds
- Analyzed data to understand tradeoffs

Analysis Methodology

- Multiple linear (and logistic) regression
  - Predict precision and performance (dependent vars) as a linear function of analysis options (independent vars)
    - Programs are indep. vars, to isolate effects of size, complexity, …
    - Also considered two-way interactions of options as indep. vars
  - Assuming a good model fit, doing this helps understand the complex space of possible options
- Visualization plotting precision vs. performance, per program, for all options
  - To see local variations, sanity-check trends

Results

- Results clarify tradeoffs among analysis options
  - Yes, convex polyhedra are more precise, but slower, than intervals; but our results say **how much**,
  - and how these tradeoffs relate to that of varying the heap model (e.g., summary objects vs. access paths)
- General: Need more empirical work
  - Research often focuses on novelty, and math
  - Need as much or more work on engineering and measurement
  - Our methodology can help show the way

Learn More

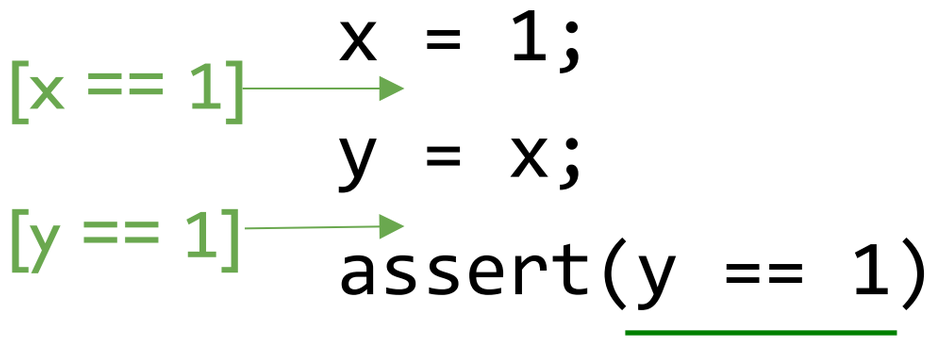
- Paper at https://arxiv.org/abs/1802.08927
  - Longer version of paper from ESOP'18
- Code at https://github.com/plum-umd/JANA

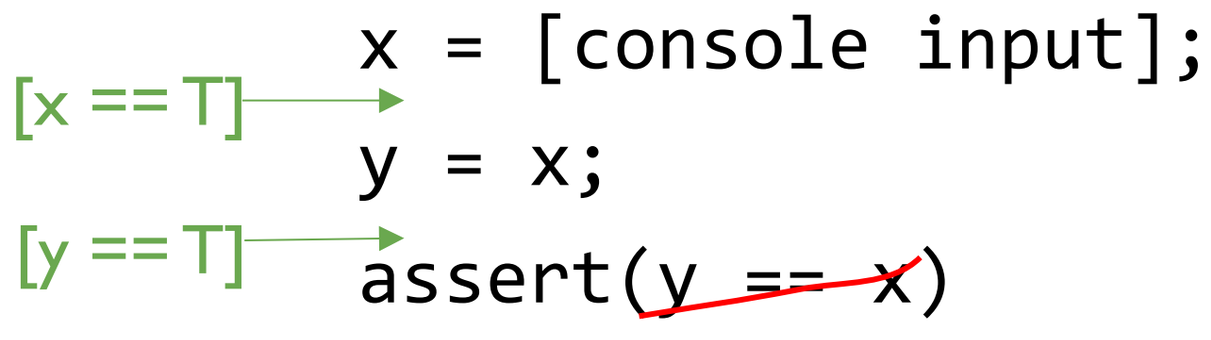
Overview: Analysis Configurations

Config. Option	Setting	Description
Numeric domain (ND)	INT	Intervals
	POL	Polyhedra
Heap abstraction (HA)	SO	Only summary objects
	AP	Only access paths
	AP+SO	Both access paths and summary objects
Abstract object representation (OR)	ALLO	Alloc-site abstraction
	CLAS	Class-based abstraction
	SMUS	Alloc-site except Strings
Inter-procedural analysis order (AO)	TD	Top-down
	BU	Bottom-up
	TD+BU	Hybrid top-down and bottom-up
Context sensitivity (CS)	CI	Context-insensitive
	1CFA	1-CFA
	1TYP	Type-sensitive

Numeric Domain

non-relational numeric domain
(e.g., intervals)



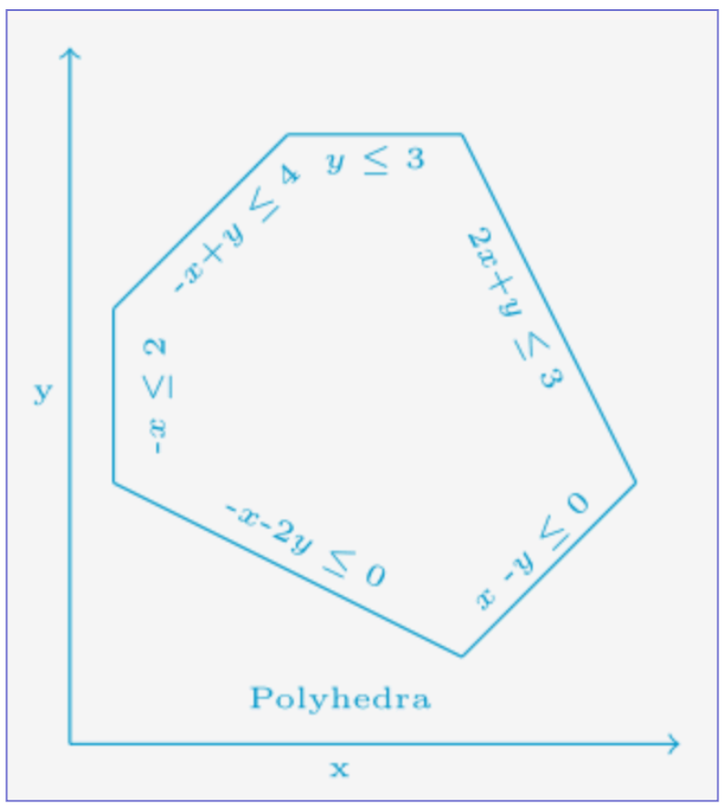


Numeric Domain

relational numeric domain
(e.g., convex polyhedra)

```
[x == 1]      x = 1;
              y = x;
[y == 1]      assert(y == 1)
```

```
[x == T]      x = [console input];
              y = x;
[x == T, y == x]   assert(y == x)
```



Tradeoff I: Numeric Domain

More Precise

Convex polyhedra vs. Intervals

More Efficient

Adding the Heap: Points-to Analysis

- Use points-to analysis to reason about aliasing
  - Doing `x.f=3` could affect `y.f` if `x` and `y` are aliases
- Compute **Pt**: *Path* ⟶ *P(Loc)* where
  - *Path*: a variable `z` or access path `x.f` and
  - *Loc* *o* represents an abstract location
    - The name *o* represents one or more run-time values
  - Hence **Pt**(p) ∩ **Pt**(q) ≠ ∅ implies p and q may alias
- Using **Pt**, we define a heap abstraction to integrate with our numeric domain

Access Paths [De and De'Souza ECOOP'12, Wei and Ryder ECOOP'14]

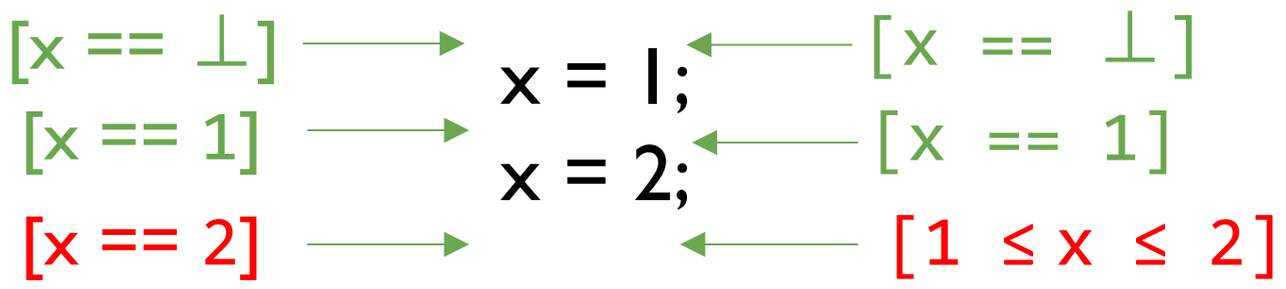
- Treat `x.f` as a numeric "variable"
  - Allow strong updates to it
  - But weakly update paths `y.f` where `y` may alias `x`

Strong update:

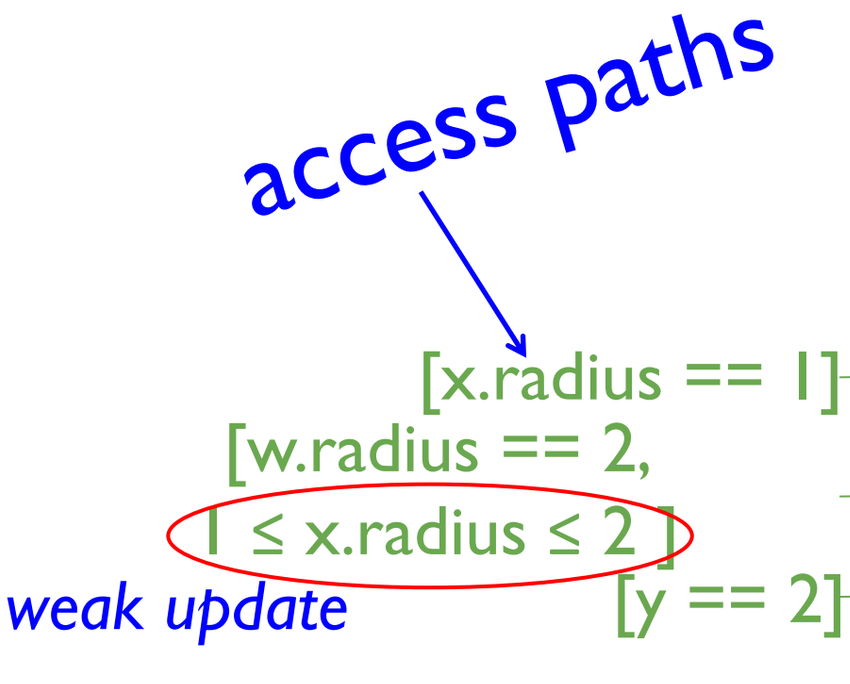
replace current abstraction

Weak update:

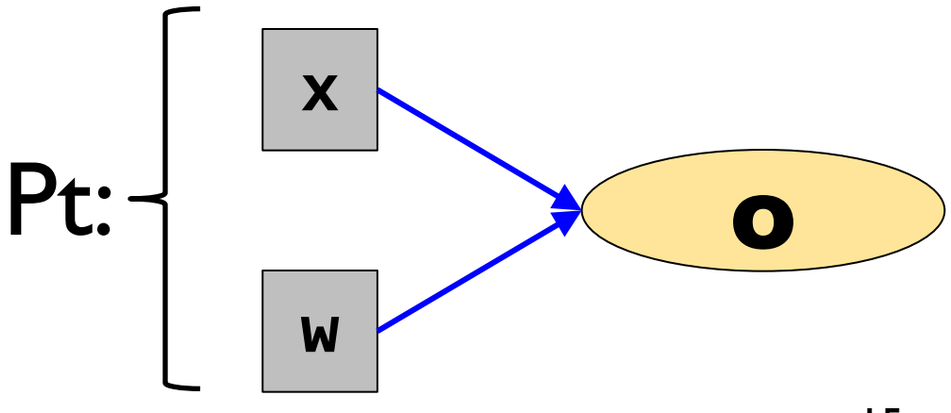
extend/overapprox current abstraction



Access Paths Example



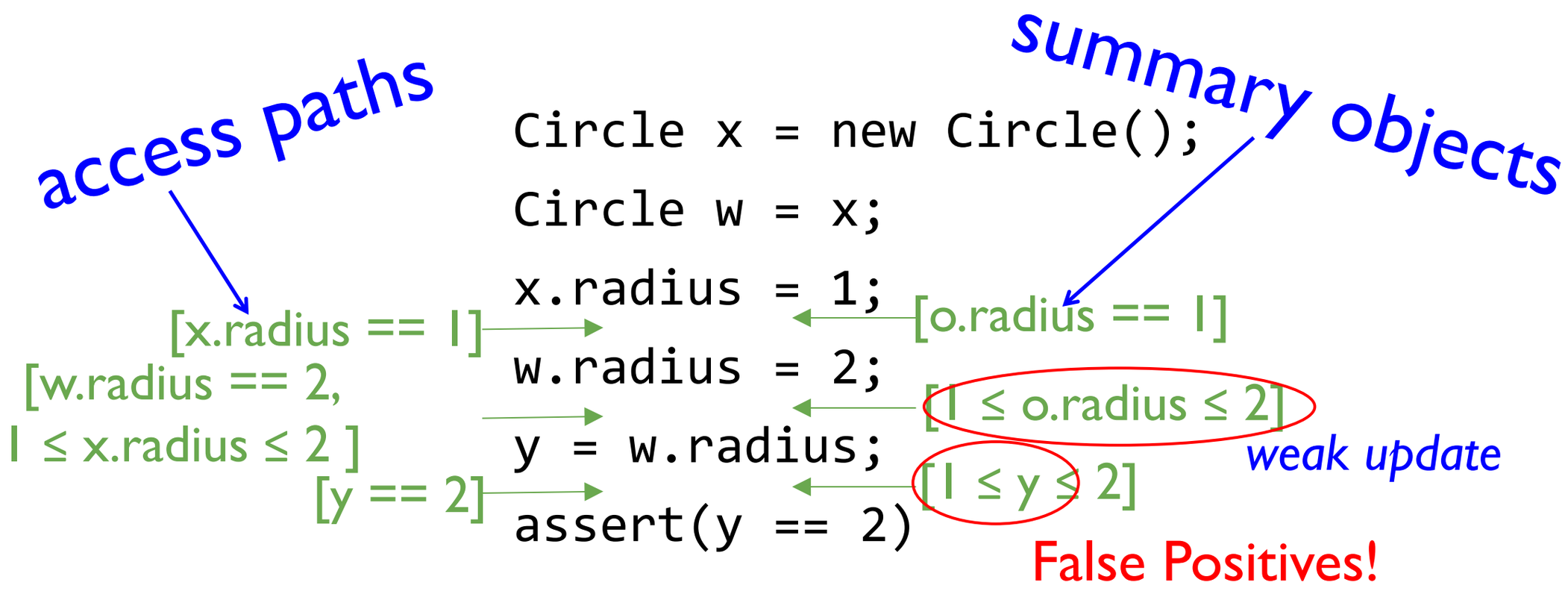

```
Circle x = new Circle();
Circle w = x;
x.radius = 1;
w.radius = 2;
y = w.radius;
assert(y == 2)
```



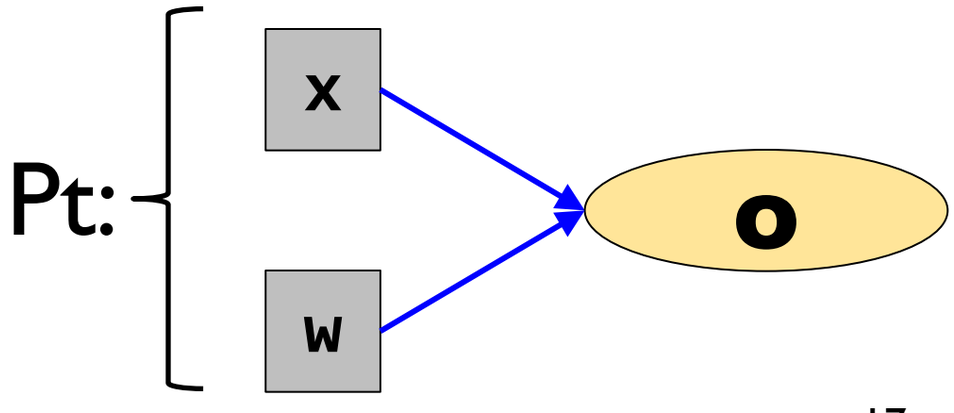
Summary Objects [Gopan et al, TACAS'04]

- Model `o.f` as a "summary" numeric object
  - When assigning to `x.f`, weakly update path `o.f` for all `o` $\in$ **Pt**(x)
- Harms and helps precision
  - No strong updates, hurts precision
  - But: Global nature of points-to analysis allows summary objects to account for effects across method calls
  - And makes expressible invariants like *o*`.f < o.g`
- Harms performance: many objects in abstract state; weak updates slow (due to joins)

Summary Objects Example

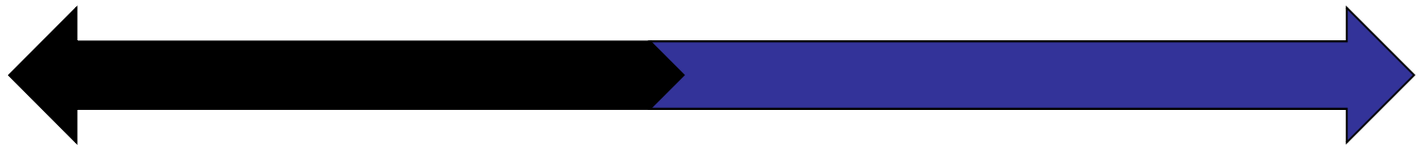


Combination of the AP and SO also possible



Tradeoff II: Heap Abstraction

More Precise



Access Paths vs. Summary Objects

More Efficient

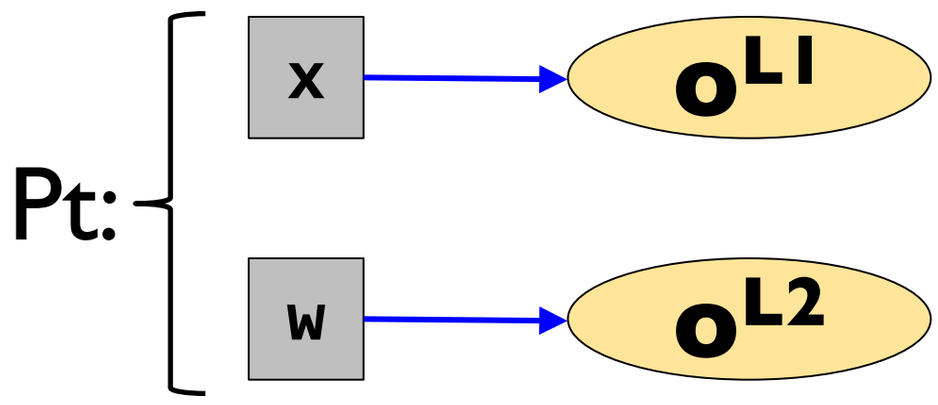
Tradeoff III: Abstract Object Representation (AOR)

Recall: **Pt**: *Path* ⟶ *P*(*Loc*).

How should we represent *Loc*, i.e., abstract objects?

Allocation site-based AOR

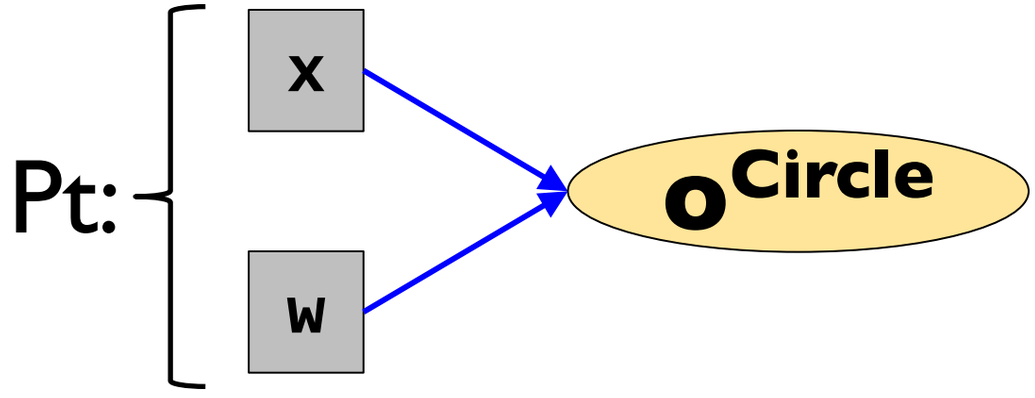
```
L1: Circle x = new Circle();
L2: Circle w = new Circle();
```



x != w

Class-based AOR

```
L1: Circle x = new Circle();
L2: Circle w = new Circle();
```



x != w

or

x == w

AOR Effect on Access Paths

```
Circle x = new Circle();
Circle w = new Circle();
x.radius = 1;
w.radius = 2;
y = x.radius;
assert(y == 1)
```

[x.radius == 1] → x.radius = 1;

[x.radius == 1] ← w.radius = 2;

Weak update to x since x ?= w

[x.radius == 1] → y = x.radius;

[1 ≤ x.radius ≤ 2] ← y = x.radius;

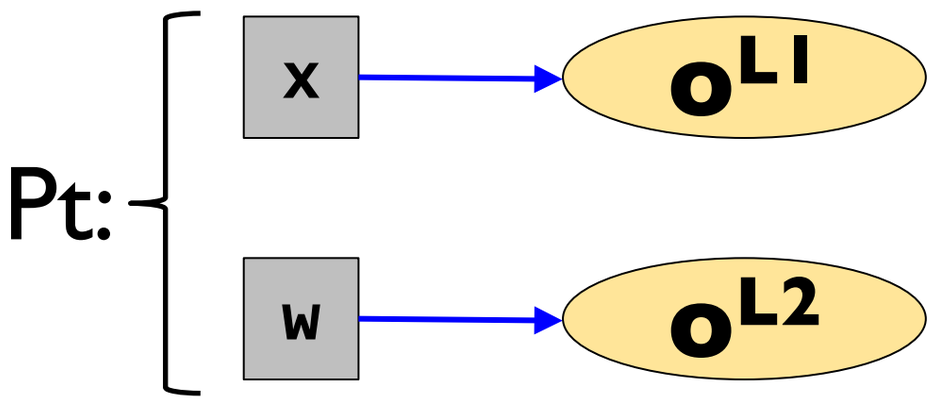
No weak update to x since x != w

[y == 1] → assert(y == 1)

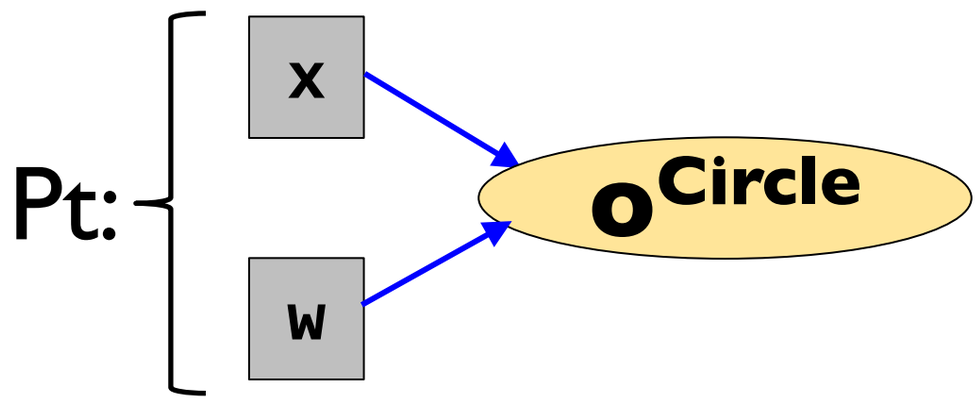
[1 ≤ y ≤ 2] ← assert(y == 1)

False Positive!

allocation-based



class-based



AOR also affects summary object representation

Tradeoff III: Abstract Object Representation

More Precise

Allocation-based vs. Smush-string vs. Class-based

More Efficient

- The smush-string option is a hybrid
  - all Java String instances use a single abstract object
  - other objects uses allocation-based approach

Tradeoff IV: Context Sensitivity

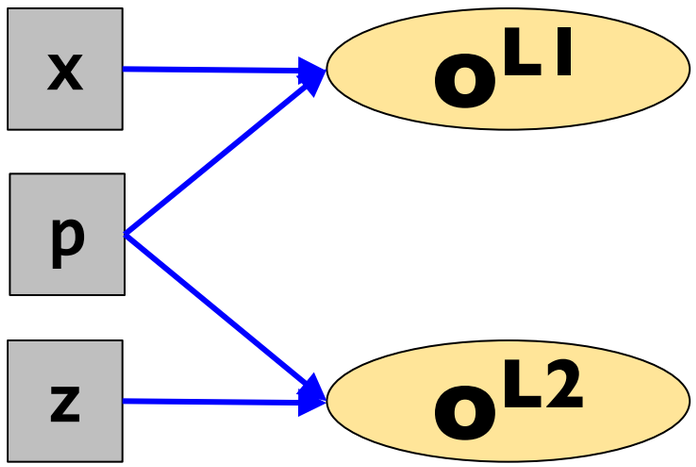
```
void foo() {
  Circle x = new Circle();
  Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  int i = bar(x);
  int j = bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

Tradeoff IV: Context Sensitivity

```
void foo() {
  L1: Circle x = new Circle();
  L2: Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  L3: int i = bar(x);
  L4: int j = bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

Tradeoff IV: Context Sensitivity

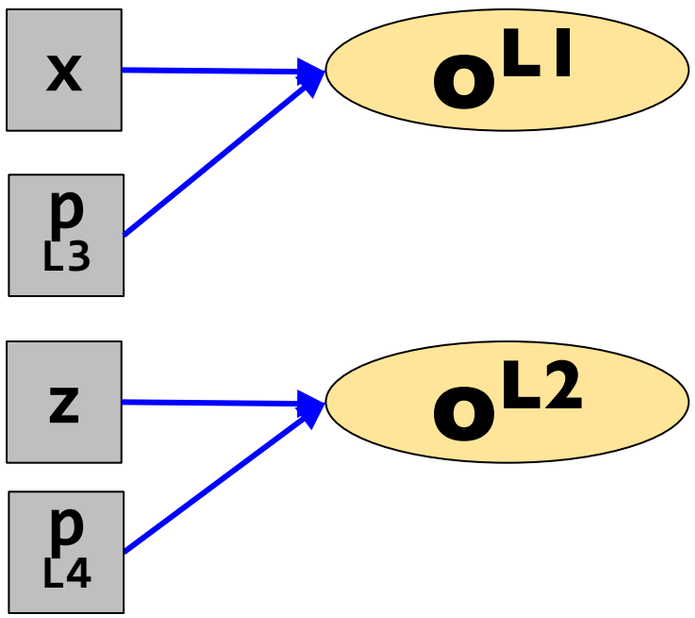
context insensitivity




```
void foo() {
  L1: Circle x = new Circle();
  L2: Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  L3: int i = bar(x);
  L4: int j = bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

Tradeoff IV: Context Sensitivity

call-site sensitivity (1CFA)




```
void foo() {
  L1: Circle x = new Circle();
  L2: Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  L3: int i = bar(x);
  L4: int j = bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

Also affects precision of call graph and the way method summaries are stored

Tradeoff IV: Context Sensitivity

```
void foo() {
  L1: Circle x = new Circle();
  L2: Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  L3: int i = bar(x);
  L4: int j = bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

[x != z]

[x.radius == 1, x != z]

[x.radius == 1, z.radius == 2]

[i == 2]

[j == 3]

call-site-sensitive

[x.radius == 1, z.radius == 2, p^{L3} != z]

[ret == 2]

Tradeoff IV: Context Sensitivity

```
void foo() {
  L1: Circle x = new Circle();
  L2: Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  L3: int i = bar(x);
  L4: int j = bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

[x != z]

[x.radius == 1, x != z]

[x.radius == 1, z.radius == 2]

[i == 2]

[j == 3]

[x != z]

[x.radius == 1, x != z]

[x.radius == 1, z.radius == 2]

[i == 2 or i ==3]

[j ==2 or j == 3]

False Positives!

call-site-sensitive

[x.radius == 1, z.radius == 2, p^{L3} != z]

[ret == 2]

[x.radius == 1, z.radius == 2, p == z or p == x]

[ret == 2 or ret ==3]

context-insensitive

Tradeoff IV: Context Sensitivity

More Precise

Context Sensitivity vs. Type Sensitivity vs. Context Insensitivity

More Efficient

- Type-sensitive analysis distinguishes the function calls based on the type of the receiver

Tradeoff V: Inter-procedural Analysis Order

- **Top-down** interprocedural analysis
  - Analyzes each method starting at main, following the call graph down; carries forward access paths
- Precise analysis of each call
  - Boosted by context sensitivity if enabled
- But may reanalyze same method many times
  - But: we cache a summary of a method's analysis, indexed by facts about its arguments
  - Reuse summary if subsequent calls' arguments' abstractions covered by those of previous analysis

Tradeoff IV: Inter-procedural Analysis Order

top-down

[x != z]

[x.radius == 1, x != z]

[x.radius == 1, z.radius == 2]

[i == 2]

[j == 3]

[p^{L3}.radius == 1]

[ret == 2]

```
void foo() {
  Circle x = new Circle();
  Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  int i = L3:bar(x);
  int j = L4:bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

Tradeoff V: Inter-procedural Analysis Order

- **Bottom-up** analysis
  - Analyzes each method starting at leaves, generating a summary that can be used at each caller
- No re-analysis, amenable to parallelization
  - Context sensitivity requires multiple summaries for each method (one per context)
- May lose precision depending on summary format
  - Convex polyhedra permit expressing relationships between arguments and return value; but not intervals
- Complicated by use of summary objects
  - Harms performance and precision

Tradeoff IV: Inter-procedural Analysis Order

top-down

```
void foo() {
  Circle x = new Circle();
  Circle z = new Circle();
  x.radius = 1;
  z.radius = 2;
  int i = L3:bar(x);
  int j = L4:bar(z);
  assert (i + j == 5);
}
int bar(Circle p) {
  return p.radius + 1;
}
```

[x != z] → (at x.radius = 1;)

[x.radius == 1, x != z] → (at z.radius = 2;)

[x.radius == 1, z.radius == 2] → (at int i = L3:bar(x);)

[i == 2] → (at int j = L4:bar(z);)

[j == 3] → (at assert (i + j == 5);)

[p^{L3}.radius == 1] → (at return p.radius + 1;)

[ret == 2] → (at })

[x != z] ← (at Circle z = new Circle();)

[x.radius == 1, x != z] ← (at x.radius = 1;)

[x.radius == 1, z.radius == 2] ← (at z.radius = 2;)

[i == 2] ← (at int i = L3:bar(x);)

[j == 3] ← (at int j = L4:bar(z);)

bottom-up

Sum(bar^{L3}) = [ret = p^{L3}.radius + 1]

Sum(bar^{L4}) = [ret = p^{L4}.radius + 1]

Tradeoff V: Inter-procedural Analysis Order

More Precise

Top-down vs. *Hybrid* Top-down+Bottom-up vs. Bottom-up

More Efficient

- Hybrid TD+BU option performs
  - bottom-up analysis for the Java library methods and
  - top-down analysis for the application code

Our tool: JANA (Java Numeric Analyzer)

https://github.com/plum-umd/JANA

- WALA Java analysis framework
  - To parse bytecode to IR
  - To perform points-to analysis, CFG construction
- Numeric domain backends
  - APRON for Intervals
  - ELINA for Convex Polyhedra
- 14K lines of Scala code

Experimental Evaluation

- RQ1: performance
  - How does the configuration affect running time?
- RQ2: precision
  - How does the configuration affect precision?
- RQ3: tradeoffs
  - How does the configuration affect the precision/performance tradeoff?

Note: a configuration is one combination of the analysis configuration options.

Evaluation: Setup

- Analysis client: array index-out-of-bound analysis
- DaCapo benchmark suite
  - most popular Java benchmark

Program	Size (# IR instructions)	# Checks
antlr	55734	1526
bloat	150197	4621
chart	167621	7965
eclipse	18938	1043
fop	33243	1337
hsqldb	19497	1020
jython	127661	4232
luindex	69027	2764
lusearch	20242	1062
pmd	116422	4402
xalan	20315	1043

Evaluation: Setup

- Analysis client: array index-out-of-bound analysis
- DaCapo benchmark suite
  - most popular Java benchmark
  - 11 real-world programs (some very large)
- 162 analysis configurations
  - vary each of the analysis options
  - 3 runs for each configuration on each program
- Timeout: 1 hour per run
  - 1 week to run the whole thing on one machine

Setup Caveats

- Results may change with different analysis client
  - Or benchmark suite
- Three trials may be insufficient
  - Average variance ((max-min)/median) of all runs: 4.2%
    - most runs differ by at most 4 mins (outlier: 32 mins, eclipse)
- Our implementation may have flaws
  - Though we used time-tested libraries and tools and tried to follow best practice

Evaluation: Data Analysis

- Multiple linear regression
  - dependent variables: performance and/or whether run times out (RQ1) and precision (RQ2)
  - independent variables
    - configuration options (and their two-way interactions)
    - benchmark program: allow us roughly factor out program-specific sources of performance or precision gain/loss (e.g., size)
- Used Akaike Information Criterion (AIC) to drop insignificant variables from the model
- Model R^2: Performance: 0.72, precision: 0.98
  - Good fit

Timeouts

(Follows advice of Arcuri and Briand (ICSE'11))

- In performance analysis, timed out runs are counted as if taking 1 hour
  - Thus indicate a lower bound of the true cost
- In precision analysis, timed out runs ignored
  - Too little data means lower statistical significance
- Separate timeout analysis in paper
  - How likely are particular options to induce timeout?
  - Teaser: Using AP rather than SO reduces timeout likelihood by 40,000 times (!)

RQ1: Performance

minutes contributed to running time

95% confidence interval

Option	Setting	Est. (#)	CI	p-value	
AO	TD	-	-	-	baseline
	BU	-1.98	[-6.3, 1.76]	0.336	delta
	TD+BU	1.97	[-1.78, 6.87]	0.364	delta
HA	AP+SO	-	-	-	baseline
	AP	-37.6	[-42.36, -32.84]	<0.001	delta
	SO	0.15	[-4.60, 4.91]	0.949	delta
CS	1TYP	-	-	-	baseline
	CI	-7.09	[-10.89, -3.28]	<0.001	delta
	1CFA	1.62	[-2.19, 5.42]	0.405	delta
OR	ALLO	-	-	-	baseline
	CLAS	-11.00	[-15.44, -6.56]	<0.001	delta
	SMUS	-7.15	[-11.59, -2.70]	0.002	delta
ND	POL	-	-	-	baseline
	INT	-16.51	[-19.56, -13.46]	<0.001	delta

Green rows are statistically significant differences from baseline ($p < 0.05$)

(Lower is better)

RQ1: Performance

- Summary objects incur a significant slowdown
  - (AP is 37.6 minutes faster)

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	BU	-1.98	[-6.3, 1.76]	0.336
	TD+BU	1.97	[-1.78, 6.87]	0.364
HA	AP+SO	-	-	-
	AP	-37.6	[-42.36, -32.84]	<0.001
	SO	0.15	[-4.60, 4.91]	0.949
CS	1TYP	-	-	-
	CI	-7.09	[-10.89, -3.28]	<0.001
	1CFA	1.62	[-2.19, 5.42]	0.405
OR	ALLO	-	-	-
	CLAS	-11.00	[-15.44, -6.56]	<0.001
	SMUS	-7.15	[-11.59, -2.70]	0.002
ND	POL	-	-	-
	INT	-16.51	[-19.56, -13.46]	<0.001

(Lower is better)

RQ1: Performance

- The polyhedral domain is slow, but not as slow as summary objects

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	BU	-1.98	[-6.3, 1.76]	0.336
	TD+BU	1.97	[-1.78, 6.87]	0.364
HA	AP+SO	-	-	-
	AP	-37.6	[-42.36, -32.84]	<0.001
	SO	0.15	[-4.60, 4.91]	0.949
CS	1TYP	-	-	-
	CI	-7.09	[-10.89, -3.28]	<0.001
	1CFA	1.62	[-2.19, 5.42]	0.405
OR	ALLO	-	-	-
	CLAS	-11.00	[-15.44, -6.56]	<0.001
	SMUS	-7.15	[-11.59, -2.70]	0.002
ND	POL	-	-	-
	INT	-16.51	[-19.56, -13.46]	<0.001

(Lower is better)

RQ1: Performance

- Heavyweight CS and OR settings hurt performance, particularly when using summary objects

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	BU	-1.98	[-6.3, 1.76]	0.336
	TD+BU	1.97	[-1.78, 6.87]	0.364
HA	AP+SO	-	-	-
	AP	-37.6	[-42.36, -32.84]	<0.001
	SO	0.15	[-4.60, 4.91]	0.949
CS	1TYP	-	-	-
	CI	-7.09	[-10.89, -3.28]	<0.001
	1CFA	1.62	[-2.19, 5.42]	0.405
OR	ALLO	-	-	-
	CLAS	-11.00	[-15.44, -6.56]	<0.001
	SMUS	-7.15	[-11.59, -2.70]	0.002
ND	POL	-	-	-
	INT	-16.51	[-19.56, -13.46]	<0.001
HA:OR	AP+SO:ALLO	-	-	-
	AP:CLAS	9.55	[5.37, 13.71]	<0.001
	AP:SMUS	6.25	[2.08, 10.42]	<0.001
	SO:SMUS	0.07	[-4.09, 4.24]	0.973
	SO:CLAS	-0.43	[-4.59, 3.73]	0.839

(Lower is b

RQ1: Performance

- Bottom-up analysis does not provide a performance advantage

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	BU	-1.98	[-6.3, 1.76]	0.336
	TD+BU	1.97	[-1.78, 6.87]	0.364
HA	AP+SO	-	-	-
	AP	-37.6	[-42.36, -32.84]	<0.001
	SO	0.15	[-4.60, 4.91]	0.949
CS	1TYP	-	-	-
	CI	-7.09	[-10.89, -3.28]	<0.001
	1CFA	1.62	[-2.19, 5.42]	0.405
OR	ALLO	-	-	-
	CLAS	-11.00	[-15.44, -6.56]	<0.001
	SMUS	-7.15	[-11.59, -2.70]	0.002
ND	POL	-	-	-
	INT	-16.51	[-19.56, -13.46]	<0.001

(Lower is better)

RQ2: Precision

indexes proved in bounds

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	TD+BU	-134.22	[-184.93, -83.50]	<0.001
	BU	-129.98	[-180.24, -79.73]	<0.001
HA	AP+SO	-	-	-
	SO	-94.46	[-166.79, -22.13]	0.011
	AP	-5.24	[-66.47, 55.99]	0.866
OR	ALLO	-	-	-
	CLAS	-90.15	[-138.80, -41.5]	<0.001
	SMUS	35.47	[-14.72, 85.67]	0.166
ND	POL	-	-	-
	INT	5.11	[-28.77, 38.99]	0.767

(Higher is better)

RQ2: Precision

- Access paths are critical to precision

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	TD+BU	-134.22	[-184.93, -83.50]	<0.001
	BU	-129.98	[-180.24, -79.73]	<0.001
HA	AP+SO	-	-	-
	SO	-94.46	[-166.79, -22.13]	0.011
	AP	-5.24	[-66.47, 55.99]	0.866
OR	ALLO	-	-	-
	CLAS	-90.15	[-138.80, -41.5]	<0.001
	SMUS	35.47	[-14.72, 85.67]	0.166
ND	POL	-	-	-
	INT	5.11	[-28.77, 38.99]	0.767

(Higher is better)

RQ2: Precision

- Bottom-up analysis harms precision overall, especially for SO (only)

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	TD+BU	-134.22	[-184.93, -83.50]	<0.001
	BU	-129.98	[-180.24, -79.73]	<0.001
HA	AP+SO	-	-	-
	SO	-94.46	[-166.79, -22.13]	0.011
	AP	-5.24	[-66.47, 55.99]	0.866
OR	ALLO	-	-	-
	CLAS	-90.15	[-138.80, -41.5]	<0.001
	SMUS	35.47	[-14.72, 85.67]	0.166
ND	POL	-	-	-
	INT	5.11	[-28.77, 38.99]	0.767

AO:HA	TD:AP+SO	-	-	-
	BU:SO	-686.79	[-741.82, -631.76]	<0.001
	TD+BU:SO	-630.99	[-687.41, -574.56]	<0.001
	TD+BU:AP	63.59	[14.71, 112.47]	0.011
	BU:AP	58.92	[11.75, 106.1]	0.014

(Higher is

RQ2: Precision

- The relational domain improves precision some

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	TD+BU	-134.22	[-184.93, -83.50]	<0.001
	BU	-129.98	[-180.24, -79.73]	<0.001
HA	AP+SO	-	-	-
	SO	-94.46	[-166.79, -22.13]	0.011
	AP	-5.24	[-66.47, 55.99]	0.866
OR	ALLO	-	-	-
	CLAS	-90.15	[-138.80, -41.5]	<0.001
	SMUS	35.47	[-14.72, 85.67]	0.166
ND	POL	-	-	-
	INT	5.11	[-28.77, 38.99]	0.767
HA:ND	AP+SO:POL	-	-	-
	AP:INT	-58.87	[-99.39, -18.35]	0.004
	SO:INT	-61.96	[-109.08, -14.84]	0.01

(Higher is better)

RQ2: Precision

- More precise OR improves precision, but not CS (not in model)

Option	Setting	Est. (#)	CI	p-value
AO	TD	-	-	-
	TD+BU	-134.22	[-184.93, -83.50]	<0.001
	BU	-129.98	[-180.24, -79.73]	<0.001
HA	AP+SO	-	-	-
	SO	-94.46	[-166.79, -22.13]	0.011
	AP	-5.24	[-66.47, 55.99]	0.866
OR	ALLO	-	-	-
	CLAS	-90.15	[-138.80, -41.5]	<0.001
	SMUS	35.47	[-14.72, 85.67]	0.166
ND	POL	-	-	-
	INT	5.11	[-28.77, 38.99]	0.767

(Higher is better)

RQ3: Tradeoffs: Best and Worst

Prog	Size	# Checks	Best Performance Time(min)	# Checks	Percent	Best Precision Time(min)	# Checks	Percent
			BU-AP-CI-CLAS-INT			TD-AP+SO-1TYP-CLAS-INT		
antlr	55734	1526	0.6	1176	77.1%	18.5	1306	85.6%
			BU-AP-CI-CLAS-INT			TD-AP-1TYP-SMUS-POL		
bloat	150197	4621	4.0	2538	54.9%	17.2	2795	60.5%
			BU-AP-CI-CLAS-INT			TD-AP-1TYP-SMUS-INT		
chart	167621	7965	3.3	5593	70.2%	7.7	5654	71.0%
			BU-AP-CI-ALLO-INT			TD-AP+SO-1TYP-SMUS-POL		
eclipse	18938	1043	0.2	896	85.9%	3.3	977	93.7%
			BU-AP-CI-CLAS-INT			TD-AP+SO-1CFA-SMUS-INT		
fop	33243	1337	0.4	998	74.6%	2.6	1137	85.0%
			BU-AP-CI-SMUS-INT			TD-AP+SO-CI-SMUS-INT		
hsqldb	19497	1020	0.3	911	89.3%	1.4	975	95.6%
			BU-AP-CI-SMUS-INT			TD-AP-1CFA-CLAS-POL		
jython	127661	4232	1.3	2667	63.0%	33.6	2919	69.0%
			BU-AP-CI-SMUS-INT			TD-AP+SO-1TYP-ALLO-INT		
luindex	69027	2764	1.8	1682	60.9%	46.8	2015	72.9%
			BU-AP-CI-CLAS-INT			TD-AP+SO-1CFA-ALLO-POL		
lusearch	20242	1062	0.2	912	85.9%	54.2	979	92.2%
			BU-AP-CI-CLAS-INT			TD-AP+SO-CI-CLAS-INT		
pmd	116422	4402	1.7	3153	71.6%	49.5	3301	75.0%
			BU-AP-CI-CLAS-INT			TD-AP+SO-1CFA-SMUS-POL		
xalan	20315	1043	0.2	912	87.4%	3.8	981	94.1%

RQ3: Tradeoffs: Best and Worst

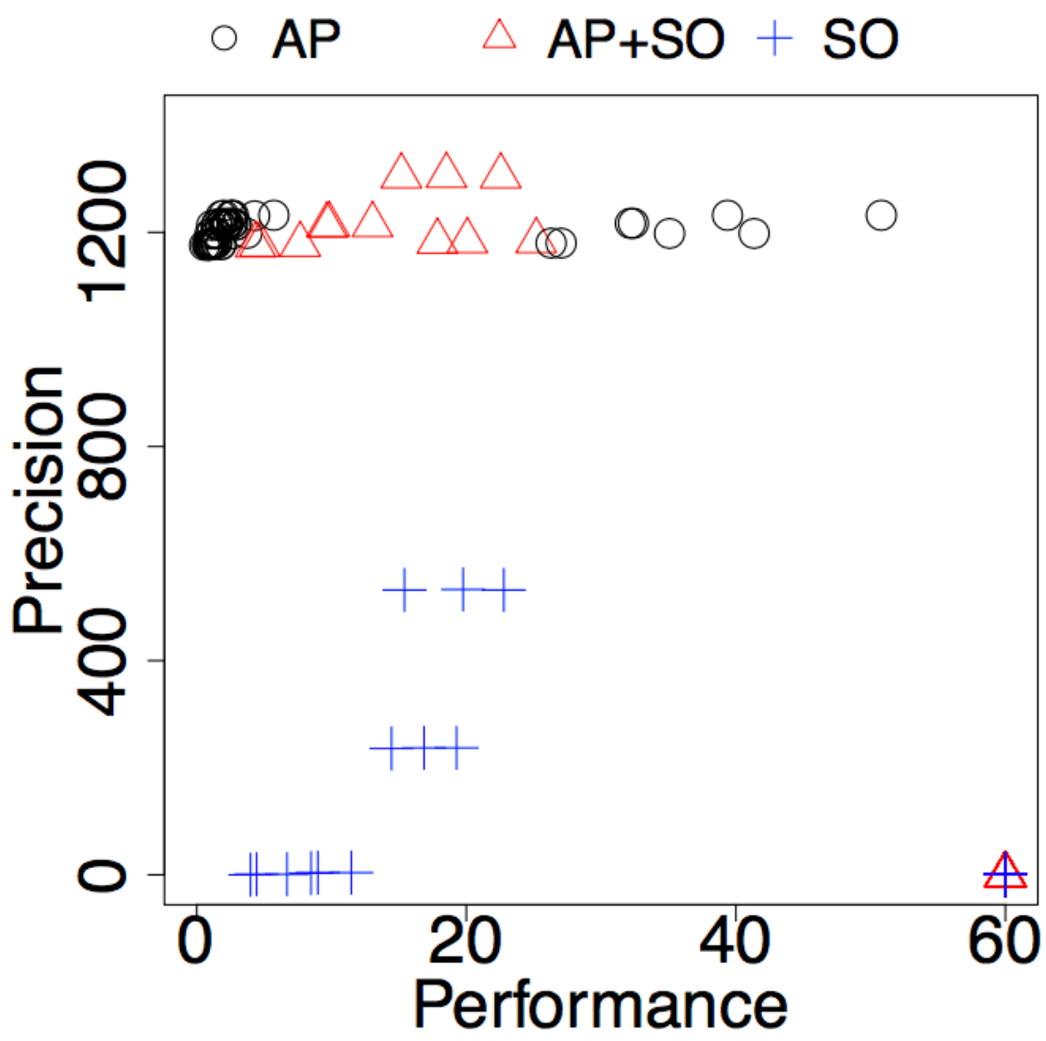
Prog	Size	# Checks	Best Performance			Best Precision		
			Time(min)	# Checks	Percent	Time(min)	# Checks	Percent
antlr	55734	1526	BU-AP-CI-CLAS-INT 0.6	1176	77.1%	TD-AP+SO-1TYP-CLAS-INT 18.5	1306	85.6%
bloat	150197	4621	BU-AP-CI-CLAS-INT 4.0	2538	54.9%	TD-AP-1TYP-SMUS-POL 17.2	2795	60.5%
chart	167621	7965	BU-AP-CI-CLAS-INT 3.3	5593	70.2%	TD-AP-1TYP-SMUS-INT 7.7	5654	71.0%
eclipse	18938	1043	BU-AP-CI-ALLO-INT 0.2	896	85.9%	TD-AP+SO-1TYP-SMUS-POL 3.3	977	93.7%
fop	33243	1337	BU-AP-CI-CLAS-INT 0.4	998	74.6%	TD-AP+SO-1CFA-SMUS-INT 2.6	1137	85.0%
hsqldb	19497	1020	BU-AP-CI-SMUS-INT 0.3	911	89.3%	TD-AP+SO-CI-SMUS-INT 1.4	975	95.6%
jython	127661	4232	BU-AP-CI-SMUS-INT 1.3	2667	63.0%	TD-AP-1CFA-CLAS-POL 33.6	2919	69.0%
luindex	69027	2764	BU-AP-CI-SMUS-INT 1.8	1682	60.9%	TD-AP+SO-1TYP-ALLO-INT 46.8	2015	72.9%
lusearch	20242	1062	BU-AP-CI-CLAS-INT 0.2	912	85.9%	TD-AP+SO-1CFA-ALLO-POL 54.2	979	92.2%
pmd	116422	4402	BU-AP-CI-CLAS-INT 1.7	3153	71.6%	TD-AP+SO-CI-CLAS-INT 49.5	3301	75.0%
xalan	20315	1043	BU-AP-CI-CLAS-INT 0.2	912	87.4%	TD-AP+SO-1CFA-SMUS-POL 3.8	981	94.1%

RQ3: Tradeoffs: Best and Worst

Prog	Size	# Checks	Best Performance			Best Precision		
			Time(min)	# Checks	Percent	Time(min)	# Checks	Percent
			BU-AP-CI-CLAS-INT			TD-AP+SO-1TYP-CLAS-INT		
antlr	55734	1526	0.6	1176	77.1%	18.5	1306	85.6%
			BU-AP-CI-CLAS-INT			TD-AP-1TYP-SMUS-POL		
bloat	150197	4621	4.0	2538	54.9%	17.2	2795	60.5%
			BU-AP-CI-CLAS-INT			TD-AP-1TYP-SMUS-INT		
chart	167621	7965	3.3	5593	70.2%	7.7	5654	71.0%
			BU-AP-CI-ALLO-INT			TD-AP+SO-1TYP-SMUS-POL		
eclipse	18938	1043	0.2	896	85.9%	3.3	977	93.7%
			BU-AP-CI-CLAS-INT			TD-AP+SO-1CFA-SMUS-INT		
fop	33243	1337	0.4	998	74.6%	2.6	1137	85.0%
			BU-AP-CI-SMUS-INT			TD-AP+SO-CI-SMUS-INT		
hsqldb	19497	1020	0.3	911	89.3%	1.4	975	95.6%
			BU-AP-CI-SMUS-INT			TD-AP-1CFA-CLAS-POL		
jython	127661	4232	1.3	2667	63.0%	33.6	2919	69.0%
			BU-AP-CI-SMUS-INT			TD-AP+SO-1TYP-ALLO-INT		
luindex	69027	2764	1.8	1682	60.9%	46.8	2015	72.9%
			BU-AP-CI-CLAS-INT			TD-AP+SO-1CFA-ALLO-POL		
lusearch	20242	1062	0.2	912	85.9%	54.2	979	92.2%
			BU-AP-CI-CLAS-INT			TD-AP+SO-CI-CLAS-INT		
pmd	116422	4402	1.7	3153	71.6%	49.5	3301	75.0%
			BU-AP-CI-CLAS-INT			TD-AP+SO-1CFA-SMUS-POL		
xalan	20315	1043	0.2	912	87.4%	3.8	981	94.1%

RQ3: Tradeoffs

Best case:
upper left
(fast and precise)



(a) antlr

Dots are plotted according to a particular option; here heap abstraction

Worst case:
lower right
(slow and imprescise)

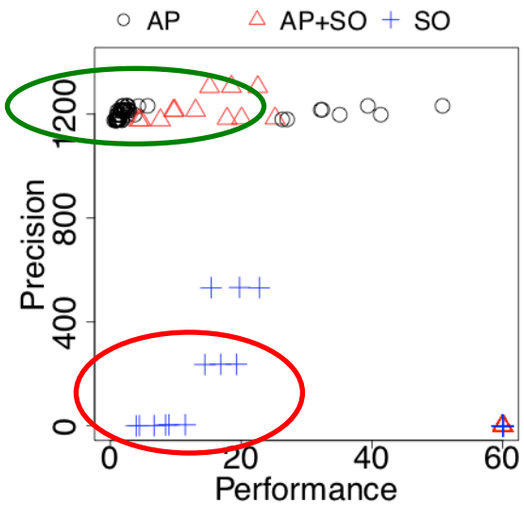
Each dot is one configuration for one program

(More graphs in paper than I can show now)

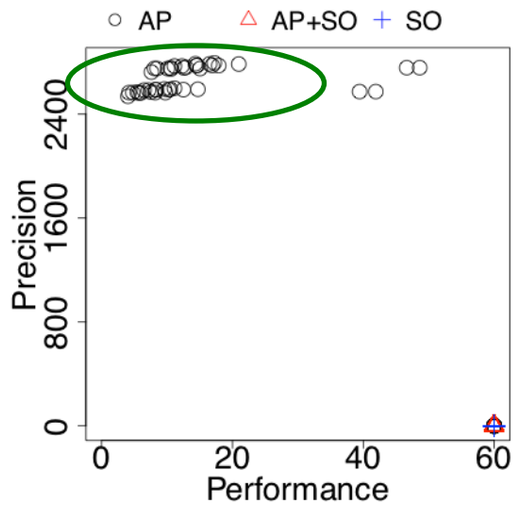
RQ3: Tradeoffs

○ AP △ AP+SO + SO

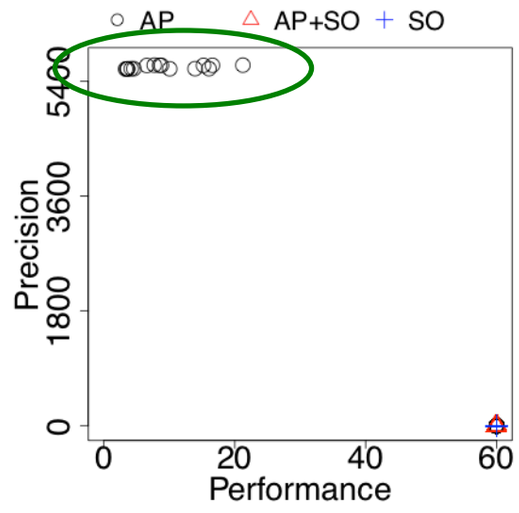
- Access paths improve precision with good performance; summary objects slow and imprecise



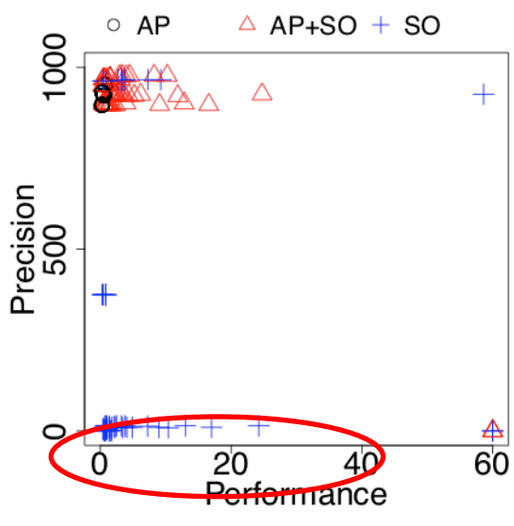
(a) antlr



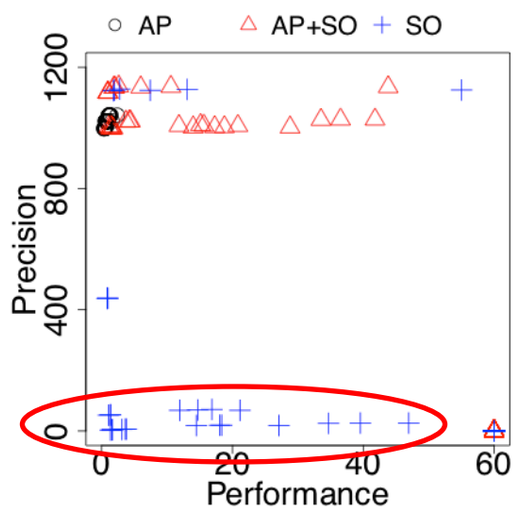
(b) bloat



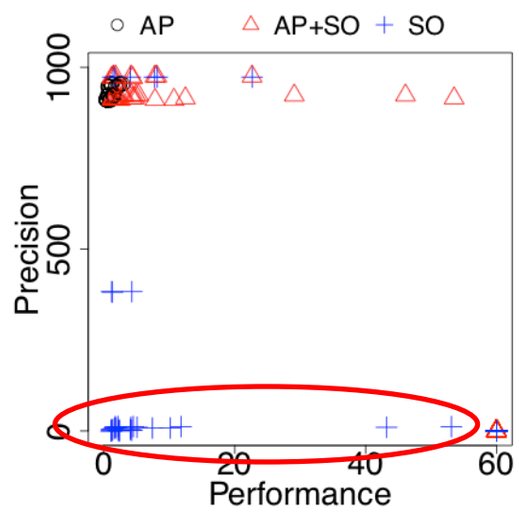
(c) chart



(d) eclipse



(e) fop

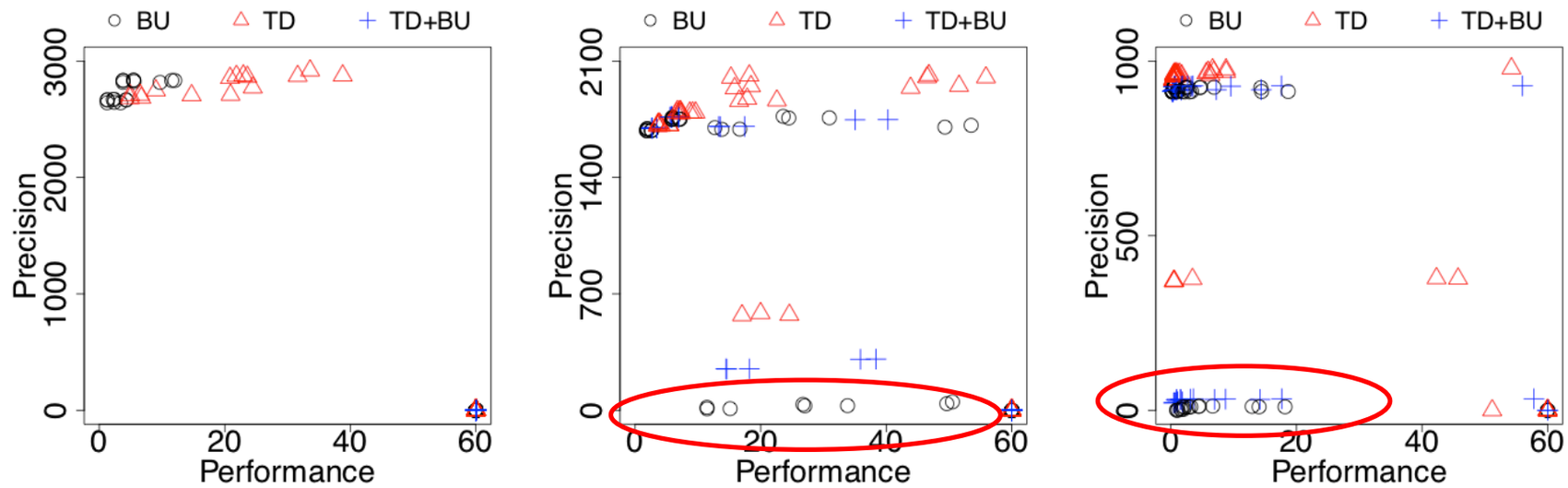


(f) hsqldb

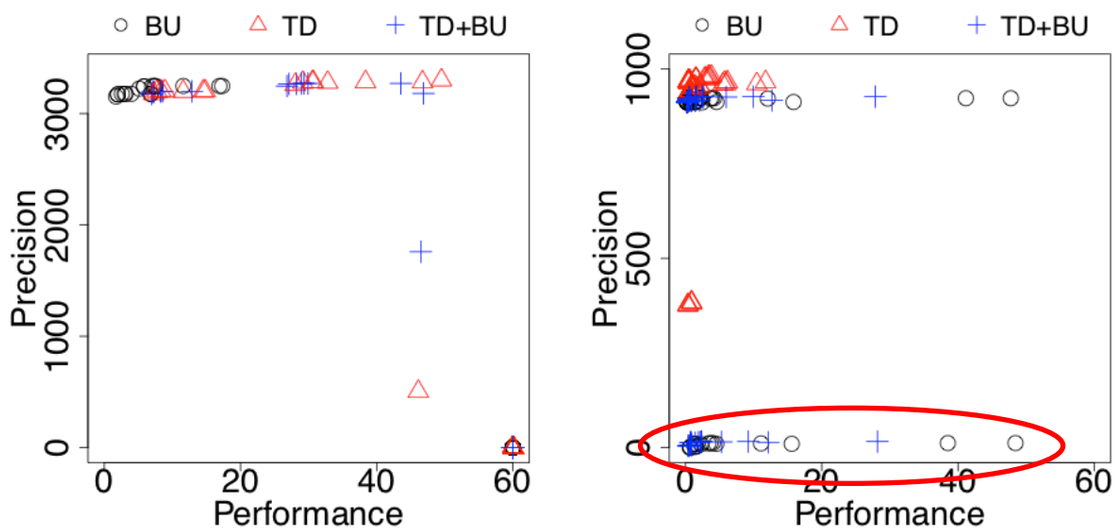
RQ3: Tradeoffs

○ BU △ TD + TD+BU

- Top-down analysis is preferred: bottom-up is less precise and does little to improve performance



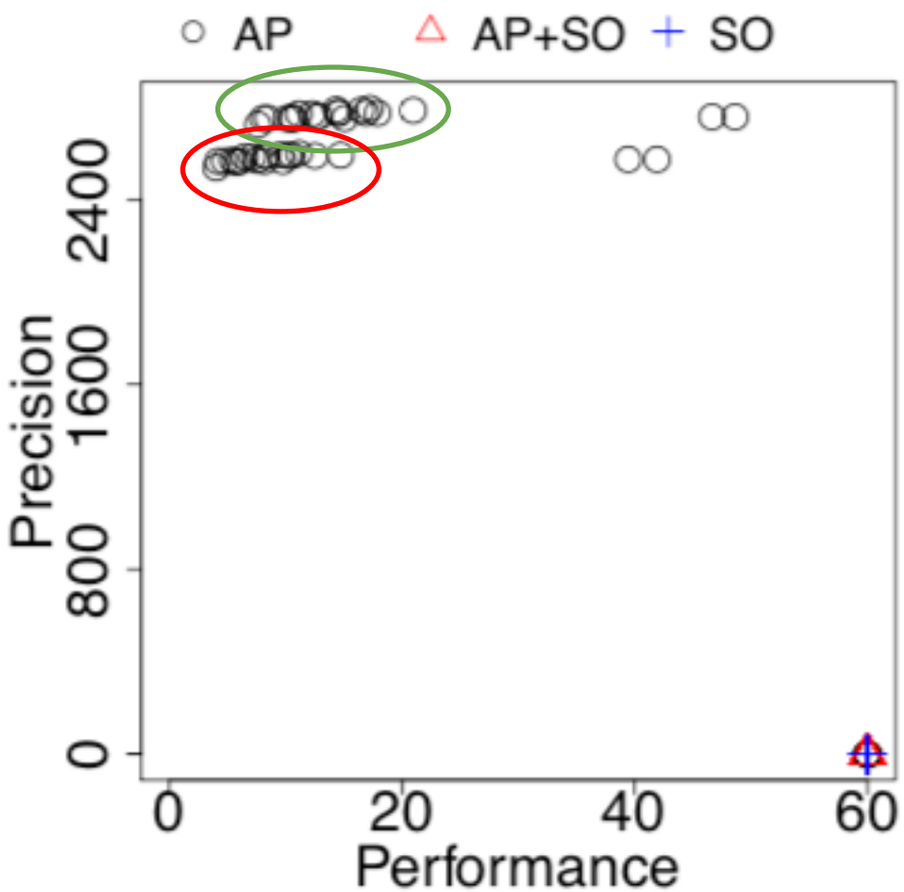
(g) jython (h) luindex (i) lusearch



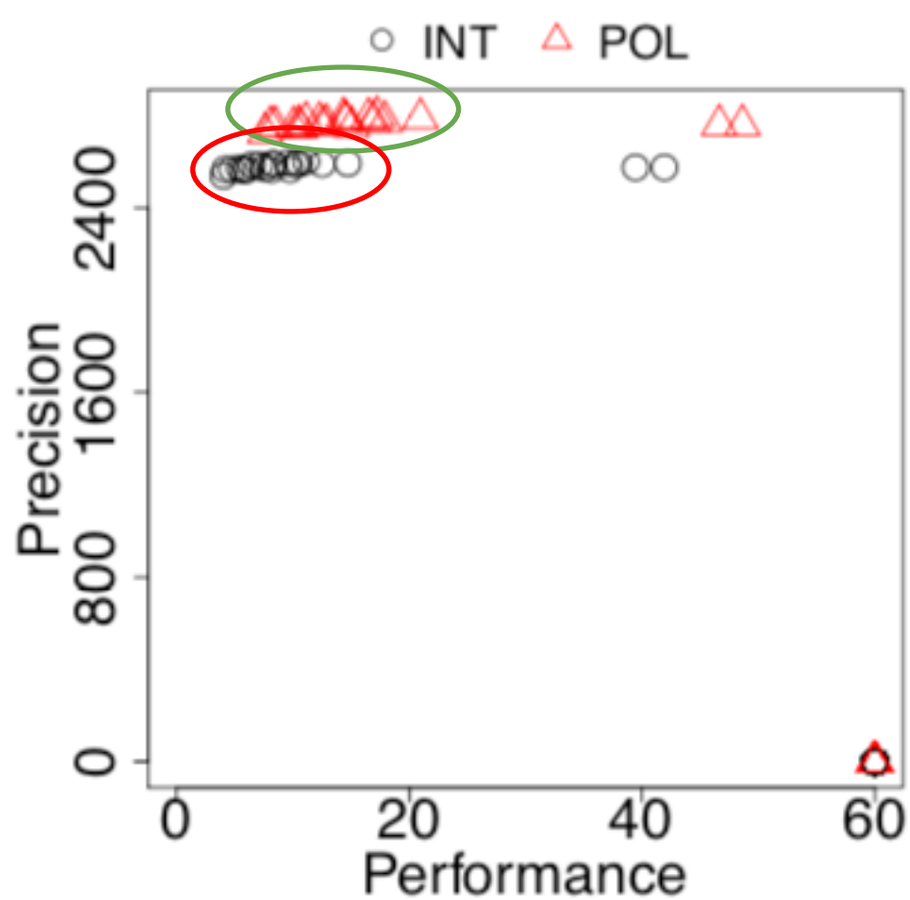
(j) pmd (k) xalan

RQ3: Tradeoffs - Interactions

- AP&POLY generally better than AP&INT



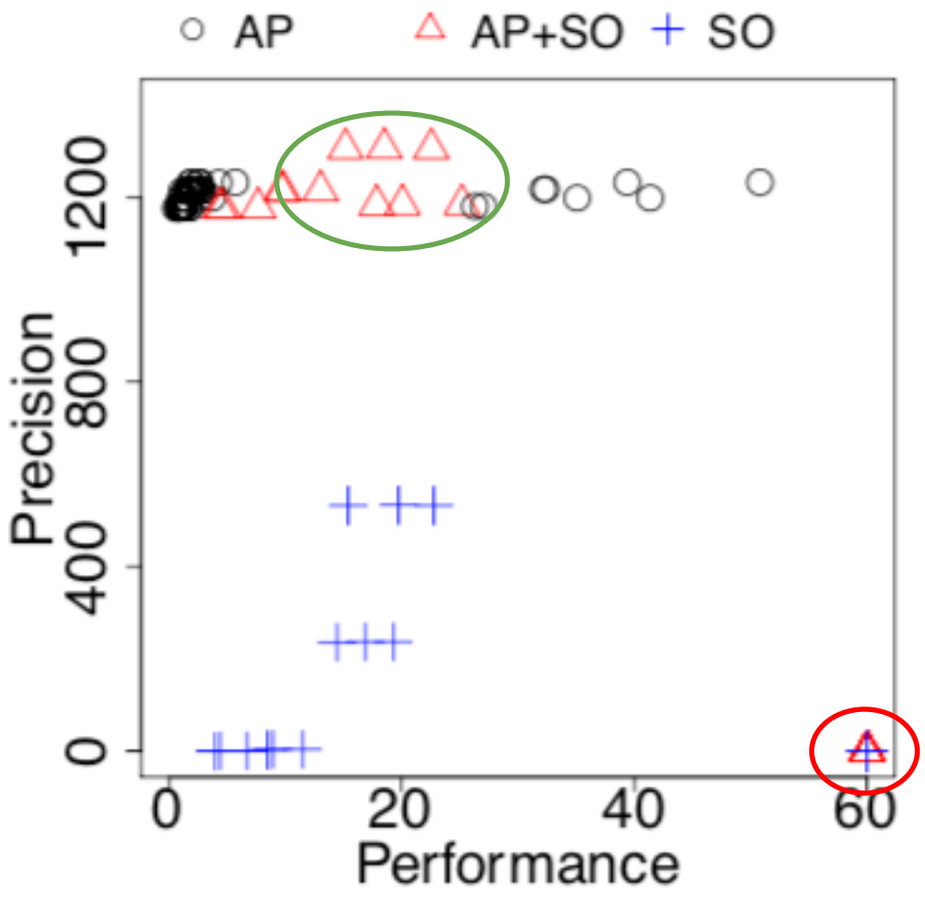
(b) bloat



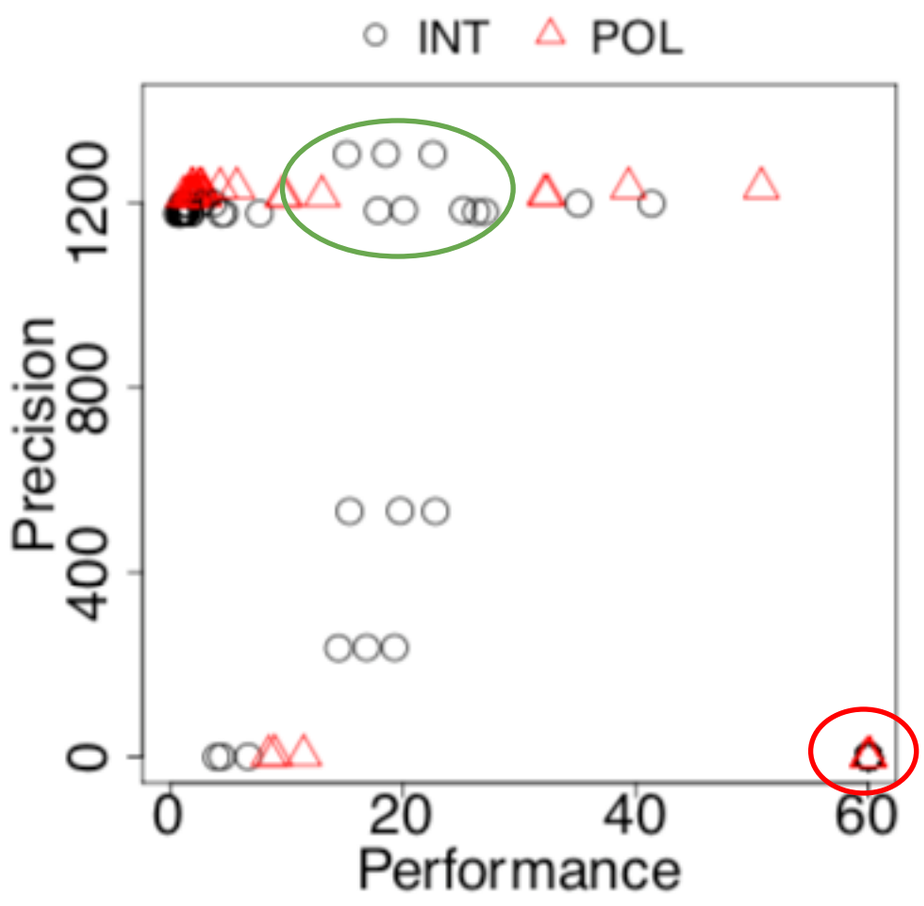
(b) bloat

RQ3: Tradeoffs - Interactions

- But AP+SO&INT better than AP+SO&POLY (fewer timeouts)



(a) antlr



(a) antlr

Results Summary

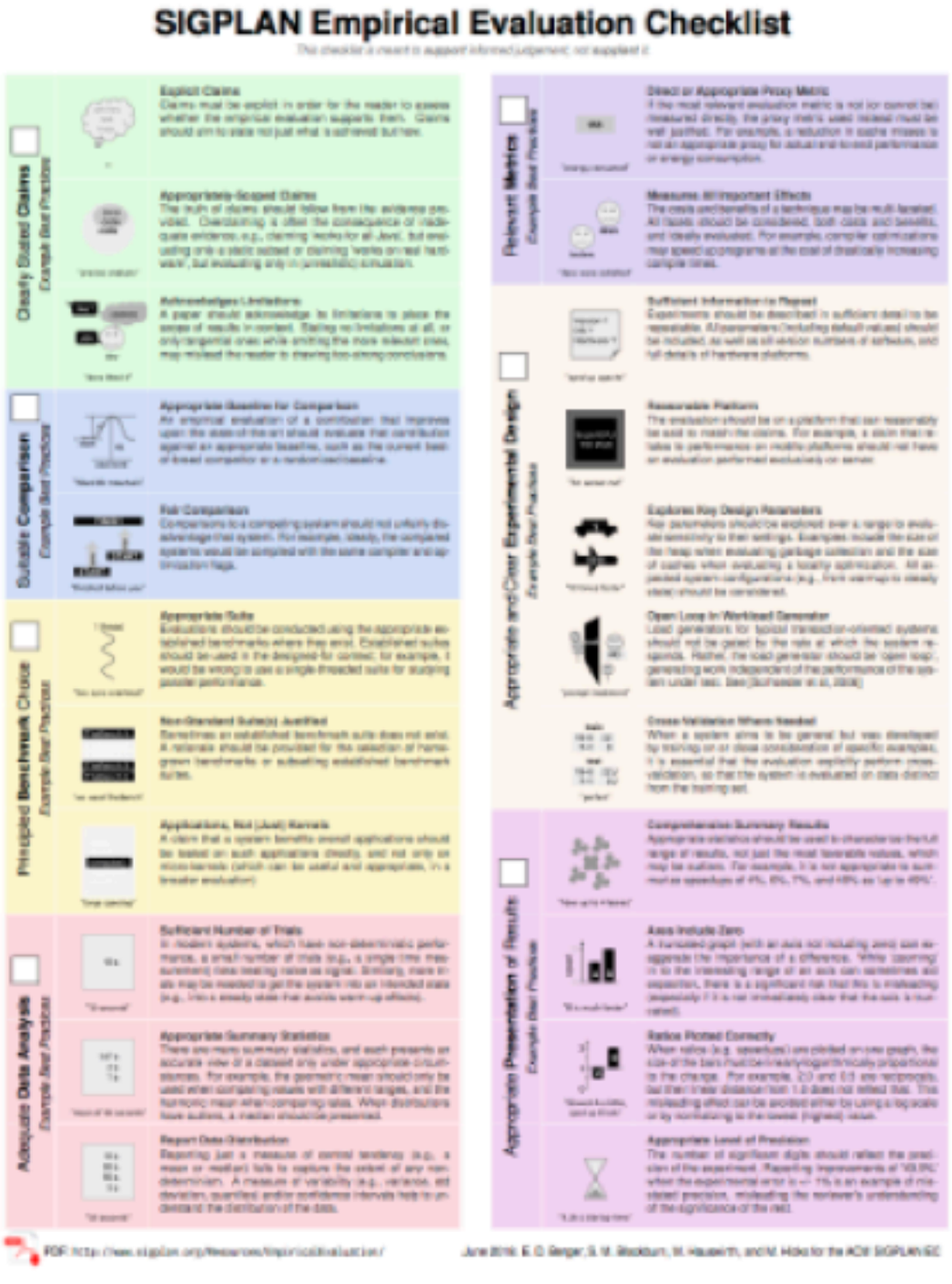
- Access paths are always a good idea
  - Add significant precision at little performance cost
- Summary objects + access paths add precision
  - But adds significant performance overhead, often resulting in timeouts
- Polyhedra improve precision
  - But time out with AP+SO abstraction
    - Intervals and AP+SO would work better
- Top-down more than precise bottom up
  - And loses little in performance

Methodology Summary

- Static analysis tools are complex, with many interacting options
- Need experimental work to understand tradeoffs
- Our work is a template for future work
  - Measure effects of various options on good benchmark
  - Use MLR to understand impact of options, generally
  - And visualization to see local effects

Empirical Evaluations: Advice

- SIGPLAN Guidelines for Empirical Evaluations
- Set up as a checklist to help design a good evaluation



http://sigplan.org/Resources/EmpiricalEvaluation/