Interactive Debugging of Datalog Programs

André Pacak and Sebastian Erdweg

JG|U Mainz

Splash Cascais 2023

Sun 22 - Fri 27 October 2023 Cascais, Portugal

Attending ▾ Tracks ▾ Organization ▾ Search Series ▾

SPLASH 2023 (series) / GPCE 2023 (series) /

GPCE 2023

New Paper Category: Generative Pearls

…is an elegant essay about generative programming

Organizing Committee

Youyou Cong — Publicity Chair
Tokyo Institute of Technology
Japan

Bernhard Rumpe — General Chair
RWTH Aachen University
Germany

Amir Shaikhha — Program Chair
University of Edinburgh
United Kingdom

Important Dates

AoE (UTC-12h)

Mon 3 Jul 2023 — new
Abstract submission

Fri 7 Jul 2023 — new
Paper submission

Wed 23 Aug 2023 — new

Interactive Debugging of Datalog Programs

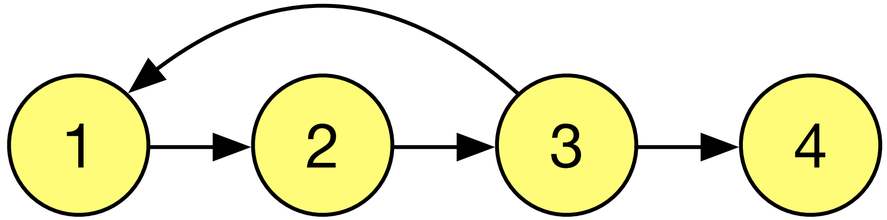
André Pacak and Sebastian Erdweg

JG|U Mainz

Datalog 101

Head ⟸ Body

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```



IDB = derived facts

```
path(1,2). path(1,3). path(1,4). path(1,1).
path(2,3). path(2,4). path(2,1). path(2,2).
path(3,4). path(3,1). path(3,2). path(3,3).
```

⟸

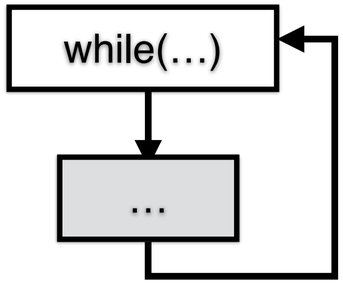
EDB = given facts

```
edge(1,2).
edge(2,3).
edge(3,4).
edge(3,1).
```

Datalog gem 1: Computable least fixpoints

```
// transitive closure
reachable(Prog, From, To) :- flow(Prog, From, To).
reachable(Prog, From, To) :- flow(Prog, From, Inter), reachable(Prog, Inter, To).
```

```
// loop detection
looping(Stm) :- reachable(Prog, Stm, Stm).
```




```
// data flow analysis
entry(Prog, Stm, X, V) :- flow(Prog, Pred, Stm), join(Pred, X, V).

exit(Prog, Stm, X, V) :- assign(Stm, X, E), eval(E, V).
exit(Prog, Stm, X, V) :- not assign(Stm, _, _), entry(Prog, Stm, X, V)
```

[ASE'16, Doop]

Datalog gem 2: Efficient incremental computing

IncA: Incremental static analysis framework

- Analyses implemented in Datalog
- Code changes fed to incremental Datalog engine

Language	Analysis	Programs	Non-inc. time	Inc. time	Speedup
Java	FindBugs	mbeddr importer (10k LoC)	n/a	7 ms	n/a
C	flow-sensitive points-to	Toyota ITC code[2](15k LoC)	5800 ms	23.3 ms	249x
C	well-formedness checks	Smart Meter (44k LoC)	209 ms	12.8 ms	16.3x
Jimple	Strong-update points-to	GTruth (9k), Gson (14k), PGSQL	6500–64300 ms	1–10 ms	650–6430 x
Jimple	String analysis	JDBC (45k), BerkleyDB (70k)	13500–20400 ms	2 ms	6500–10000x
Jimple	Constant propagation	antlr (22k), emma (26k)	5000–23000 ms	1–3 ms	1600–7600x
Jimple	Interval analysis	pmd (61k), ant (105k)	3000–23000 ms	1–6 ms	500–3800x

[ASE'16, OOPSLA'18, PLDI'21, GPCE'22]

Increasing Datalog Complexity => Need Better Dev Tools

Answer questions from users and developers:

Why does variable x point to heap location a?

```
?- varpointsto(“x”,a)
```

Why does y not point to a?

```
?- varpointsto(“y”,a)
```

What locations does z point to?

```
?- varpointsto(“z”,A)
```

Are there alternative derivations?

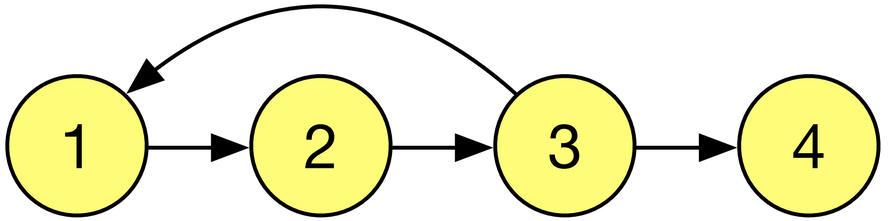
```
?- varpointsto(“x”,a)
```

```
var x = new Object // loc a
var y = new Object // loc b
var z = new Object // loc c

if (x == null) {
  y = x
}
if (Random.nextBool()) {
  z = x
} else {
  z = y
}
x = z
```

Bottom-up semantics

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```



iter = 0	iter = 1	iter = 2	iter = 3
edge(1,2).	path(1,2).	path(1,3).	path(1,4).
edge(2,3).	path(2,3).	path(2,4).	path(1,1).
edge(3,4).	path(3,4).	path(2,1).	path(2,2).
edge(3,1).	path(3,1).	path(3,2).	path(3,3).

BOTTOM-UP BEATS TOP-DOWN FOR DATALOG

Jeffrey D. Ullman
Stanford University

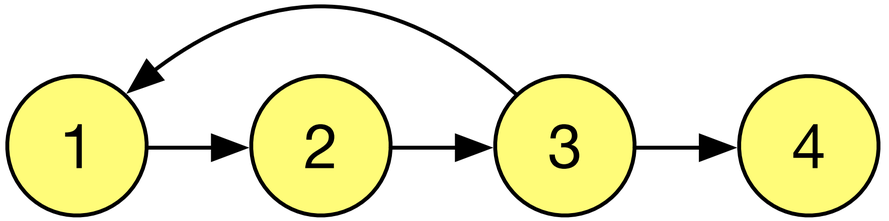
ABSTRACT

We show that for any safe datalog program $\mathcal{P}_1$ and any query Q (predicate of $\mathcal{P}_1$ with some bound arguments), there is another safe datalog program $\mathcal{P}_2$ that produces the answer to Q and takes no more time when evaluated

2. Safety of rules, that is, every variable in the head appears in some ordinary (not a built-in arithmetic comparison) subgoal.
3. Semi-naive evaluation, that is, differential computation of the least fixed point.

Bottom-up semantics

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```



iter = 0	iter = 1	iter = 2	iter = 3
edge(1,2).	path(1,2).	path(1,3).	path(1,4).
edge(2,3).	path(2,3).	path(2,4).	path(1,1).
edge(3,4).	path(3,4).	path(2,1).	path(2,2).
edge(3,1).	path(3,1).	path(3,2).	path(3,3).

Why is there a path from 3 to 3

```
?- path(3,3)
```

Why is there no path(4,1)

```
?- path(4,1)
```

What paths start at 3 and why

```
?- path(3,Y)
```

Are there alternative derivations for

```
?- path(3,4)
```

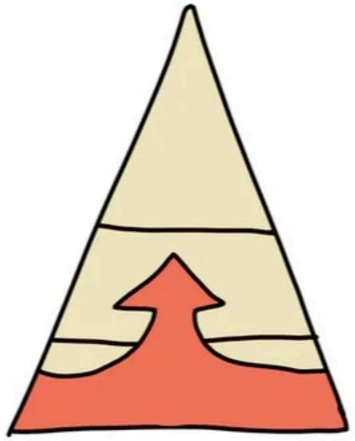
Provenance-based debugging

$$\mathrm{R2}\ \frac{\mathrm{edge}(3,1) \qquad \mathrm{R2}\ \dfrac{\mathrm{edge}(1,2) \qquad \mathrm{R1}\ \dfrac{\mathrm{edge}(2,3)}{\mathrm{path}(2,3)}}{\mathrm{path}(1,3)}}{\mathrm{path}(3,3)}$$

David Zhao, Pavle Subotic, Bernhard Scholz:
Debugging Large-scale Datalog: A Scalable Provenance Evaluation Strategy.

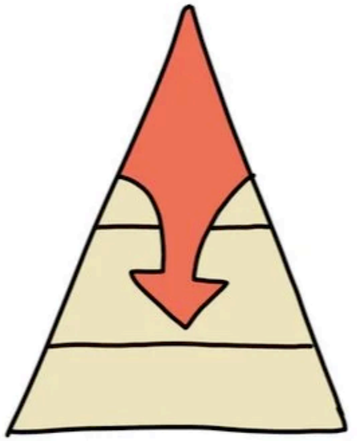
Bottom-up semantics — forward chaining

- Starts with the given facts
- Tries to derive new facts by applying rules
- Continues until no new facts can be found



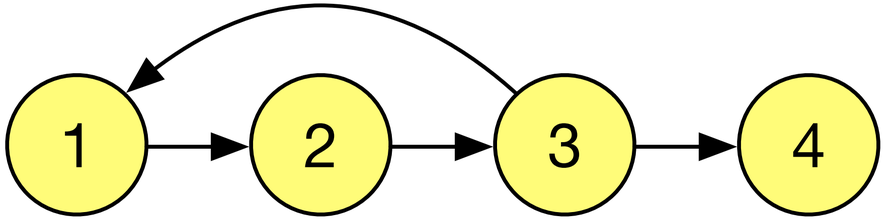
Top-down semantics — backward chaining

- Starts with a query and operates demand-driven
- Tries to answer that query by applying rules
- Continues until query is fully answered
- Yields same result as bottom-up semantics
- Very similar to standard programming experience



Top-down stepping = Interactive debugging

Why is there a path from 3 to 3




```
[X=3,Y=3]      [X=3,Y=3]
path(X,Y) :- edge(X,Y).   [X=3,Y=3]
                          {[X=3,Y=3,Z=4],
                           [X=3,Y=3,Z=1]}
[X=3,Y=3]      [X=3,Y=3]
path(X,Y) :- edge(X,Z), path(Z,Y).  [X=3,Y=3,Z=1]
```

```
{[X=4,Y=3],
 [X=1,Y=3]}
path(X,Y) :- edge(X,Y).  {[X=4,Y=3],
                          [X=1,Y=3]}
{[X=4,Y=3],   {[X=4,Y=3],
 [X=1,Y=3]}    [X=1,Y=3]}{[X=1,Y=3,Z=2]}
path(X,Y) :- edge(X,Z), path(Z,Y).
```

```
[X=1,Y=3,Z=2]
```

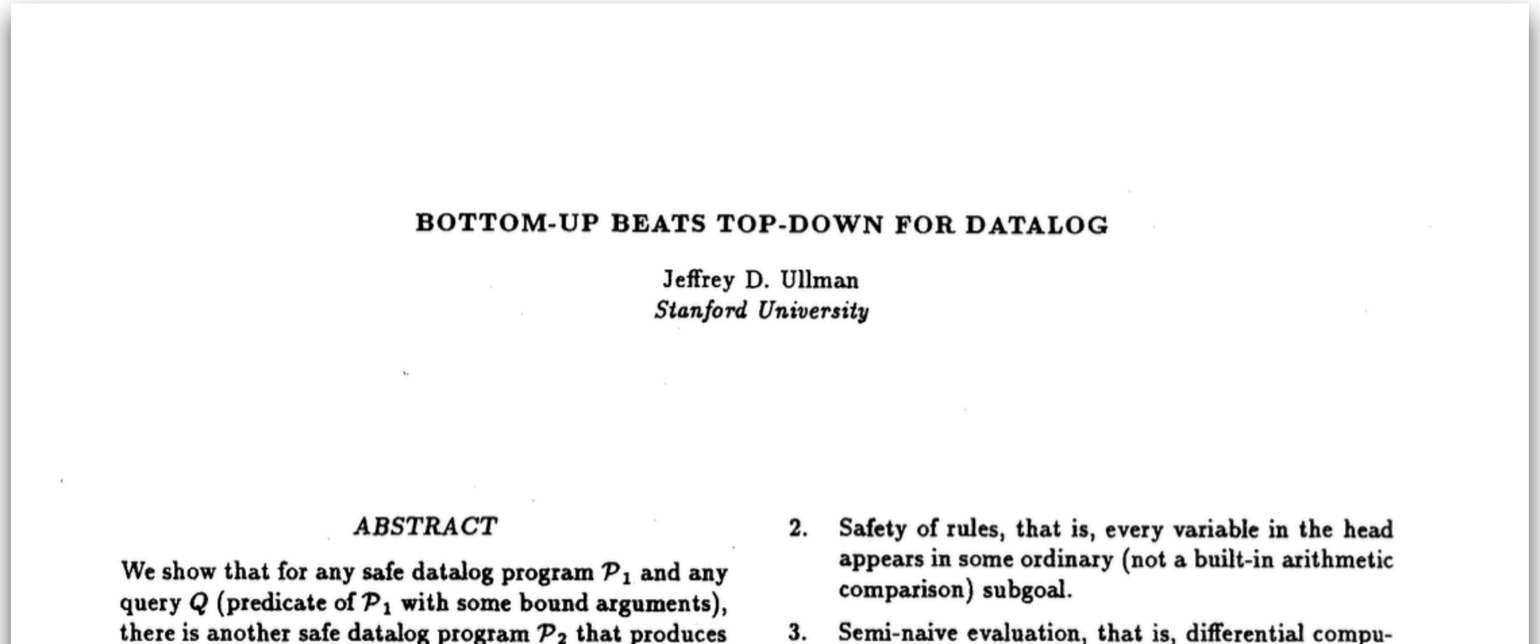
Top-down stepping = Interactive debugging

Problem 1: No top-down SOS exists for Datalog

- Existing algorithms: Query-Subquery (QSQ) Iterative and Recursive
- Beware: Lots of literature presents incomplete QSQ algorithms

Problem 2: Top-down performance is poor

- Semi-naïve evaluation and n-ary joins beat top-down
- (Virtually) all serious Datalog implementations use bottom-up



BOTTOM-UP BEATS TOP-DOWN FOR DATALOG

Jeffrey D. Ullman
Stanford University

ABSTRACT

We show that for any safe datalog program $\mathcal{P}_1$ and any query Q (predicate of $\mathcal{P}_1$ with some bound arguments), there is another safe datalog program $\mathcal{P}_2$ that produces

2. Safety of rules, that is, every variable in the head appears in some ordinary (not a built-in arithmetic comparison) subgoal.
3. Semi-naive evaluation, that is, differential compu-

Datalog syntax

(Datalog programs)	D	$::= \overline{r}, \overline{f}$
(rules)	r	$::= p(\overline{X}) :- \overline{a}.$
(atoms)	a	$::= p^s(\overline{t}) \mid \text{edb}\ p^s(\overline{t}) \mid t = t$
(signs)	s	$::= + \mid -$
(terms)	t	$::= c \mid X$
(facts)	f	$::= p(\overline{c}).$
(constants)	c	
(variables)	X	

Reduction relations

(rule reduction)	$v \vdash r \rightarrow^R r \dashv v$
(atom reduction)	$v \vdash a \rightarrow^A Q$
(query reduction)	$Q \rightarrow^Q Q$

(queries) $Q ::= \mathbf{sq}(p^s(\overline{t}), v, v, v, r \vee \ldots \vee r) \mid v^s$

- original predicate call
- query arguments
- partial subquery results
- bindings of current rule
- remaining rules

Top-down Datalog SOS

(rule reduction) $v \vdash r \rightarrow^{R} r \dashv v$

$$\text{R-Step}\ \frac{v_{sup} \vdash a \rightarrow^{A} a'}{v_{sup} \vdash p(\overline{X}) :- a, \overline{as}. \rightarrow^{R} p(\overline{X}) :- a', \overline{as}. \dashv v_{sup}}$$

$$\text{R-Merge}\ \frac{}{v_{sup} \vdash p(\overline{X}) :- v^{s}, \overline{as}. \rightarrow^{R} p(\overline{X}) :- \overline{as}. \dashv merge(v_{sup}, v^{s})}$$

$$\text{R-Result}\ \frac{}{v_{sup} \vdash p(\overline{X}) :- \epsilon. \rightarrow^{R} \Pi_{\overline{X}}(v_{sup}) \dashv v_{sup}}$$

Top-down Datalog SOS

(atom reduction) $v \vdash a \rightarrow^{A} Q$

$$\text{A-Into}\ \frac{v_a = eval(cols(p), \bar{t}, v_{sup}) \qquad (\Gamma', v'_a) = pushQuery(\Gamma, p, v_a) \qquad v'_a \text{ not empty}}{v_{sup} \vdash p^s(\bar{t}) \rightarrow^{A} \mathbf{sq}(p^s(\bar{t}), v'_a, \text{table}(cols(p), \emptyset), v'_a, rules(p))}\ \Gamma := \Gamma'$$

Top-down Datalog SOS

(query reduction) $Q \rightarrow^{Q} Q$

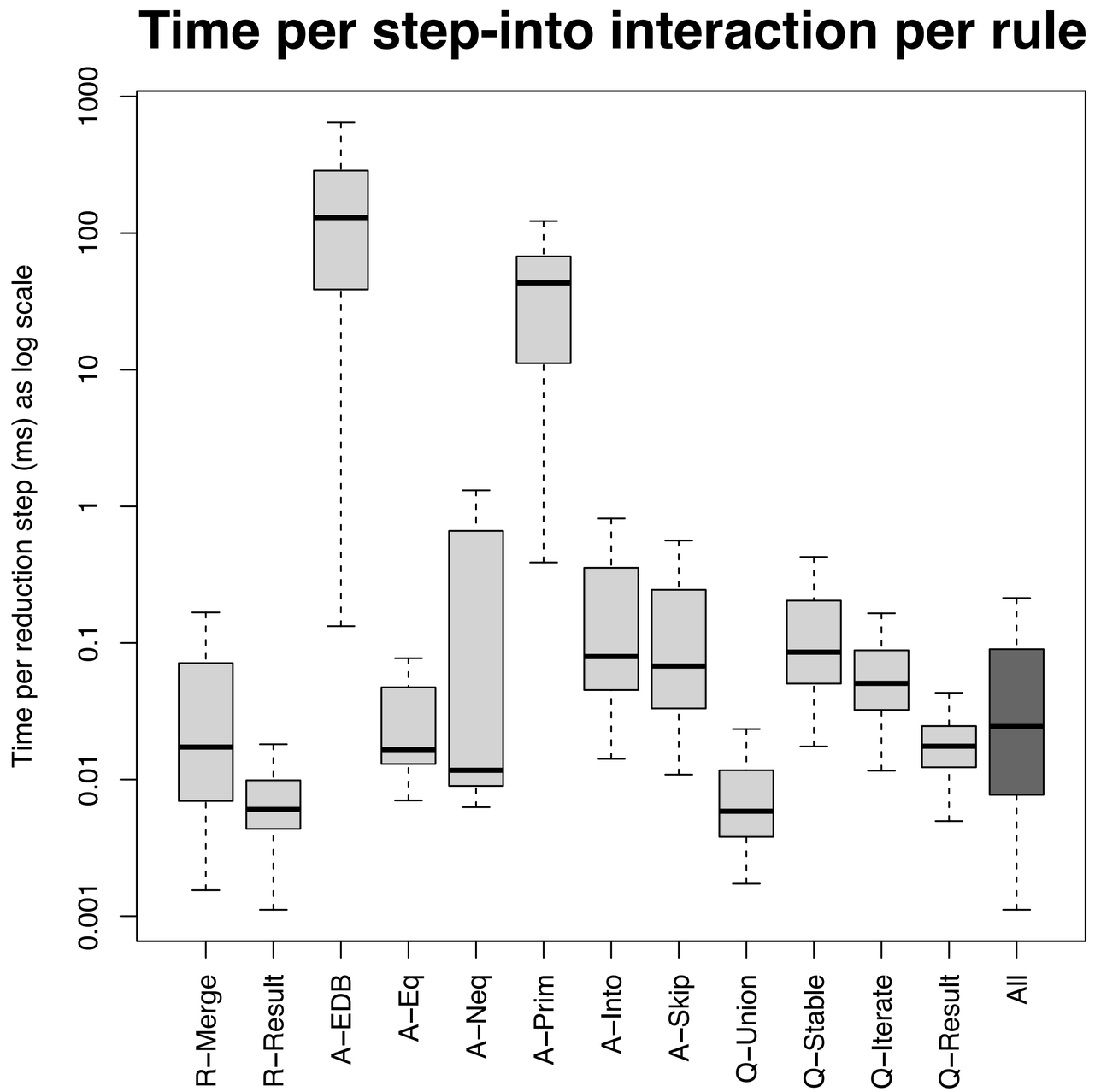
$$\text{Q-Iterate} \quad \frac{v_r \not\subseteq IDB(p) \qquad IDB' = IDB \uplus p \mapsto (v_r \cup IDB(p))}{\mathbf{sq}(p^s(\bar{t}), v_q, v_r, v_{sup}, \epsilon) \rightarrow^{Q} \mathbf{sq}(p^s(\bar{t}), v_q, \text{table}(cols(p), \emptyset), v_q, rules(p))} \; IDB := IDB'$$

$$\text{Q-Stable} \quad \frac{v_r \subseteq IDB(p) \qquad (\Gamma', v_a) = popQuery(\Gamma, p, v_q)}{\mathbf{sq}(p^s(\bar{t}), v_q, v_r, v_{sup}, \epsilon) \rightarrow^{Q} \rho_{\bar{t}/cols(p)}(IDB(p) \bowtie v_a)^s} \; \Gamma := \Gamma'$$

Implementation and Performance

Implemented in IncA

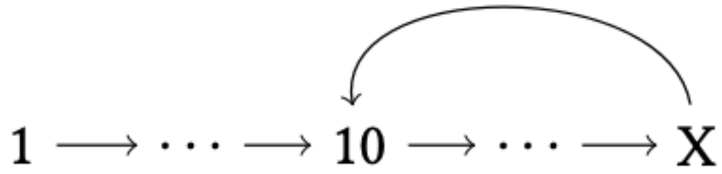
Evaluated on Doop
- ?- methLookup(C, “accept”)
- ?- supertypes({“C”,”D”}, S)
- ?- varPointsTo(“x”, a)



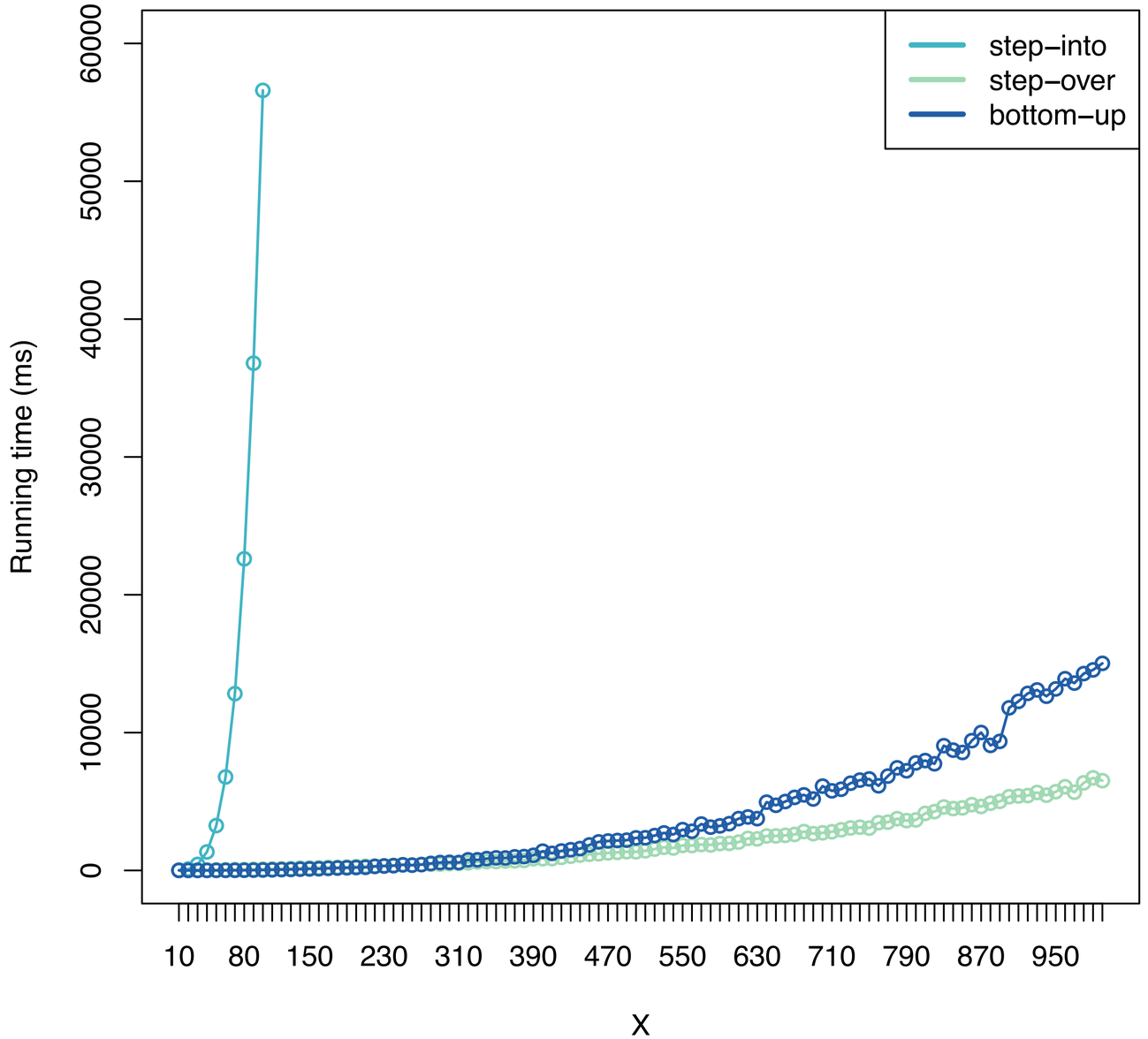
Performance problem

Simulate step-over and resume through repeated step-into?

Evaluated on synthetic cyclic graphs, computing path relation



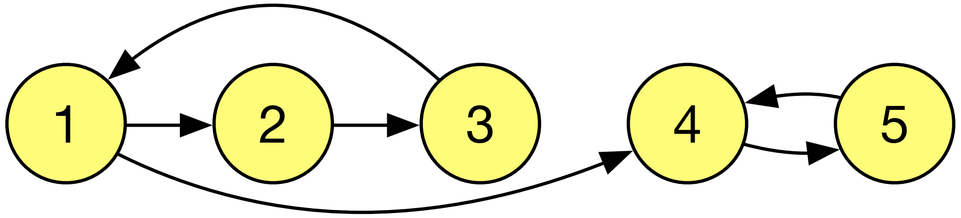
Running time for different graph sizes



Hybrid semantics

$$\text{A-Over}\quad \frac{v_a = eval(cols(p), \bar{t}, v_{sup})}{v_{sup} \vdash p^s(\bar{t}) \;\longrightarrow^A\; \rho_{\bar{t}/cols(p)}(BU(p) \bowtie v_a)^s}$$

Top-down stepping = Interactive debugging

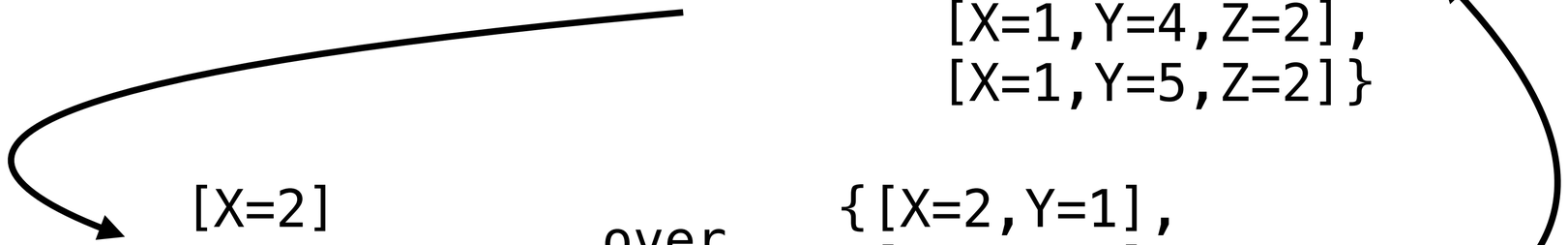



```
[X=1]          [X=1]
path(X,Y) :- edge(X,Y).   [X=1,Y=2]
```



```
[X=1]          [X=1]          [X=1,Z=2]
path(X,Y) :- edge(X,Z), path(Z,Y).
```

```
{[X=1,Y=1,Z=2],
 [X=1,Y=2,Z=2],
 [X=1,Y=3,Z=2],
 [X=1,Y=4,Z=2],
 [X=1,Y=5,Z=2]}
```



```
[X=2]
path(X,Y)
```

over

```
{[X=2,Y=1],
 [X=2,Y=2],
 [X=2,Y=3],
 [X=2,Y=4],
 [X=2,Y=5]}
```

Hybrid semantics

$$\text{A-Over}\quad \frac{v_a = eval(cols(p), \bar{t}, v_{sup})}{v_{sup} \vdash p^s(\bar{t}) \;\rightarrow^A\; \rho_{\bar{t}/cols(p)}(BU(p) \bowtie v_a)^s}$$

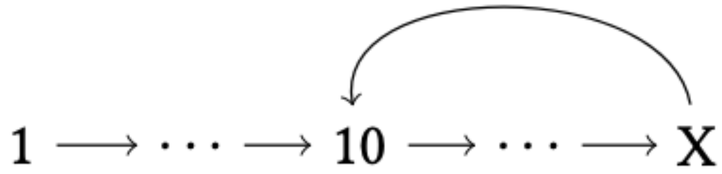
$$\text{A-OverRecursive}\quad \frac{recursiveCall(p^s(\bar{t}))\qquad v_a = eval(cols(p), \bar{t}, v_{sup})}{v_{sup} \vdash p(\bar{t})^s \;\rightarrow^A\; \rho_{\bar{t}/cols(p)}(BU(p,\Gamma) \bowtie v_a \;\cup\; IDB(p) \bowtie v_a)^s}$$

BU(p,Γ) = incrementally maintained, blacklist-aware bottom-up database

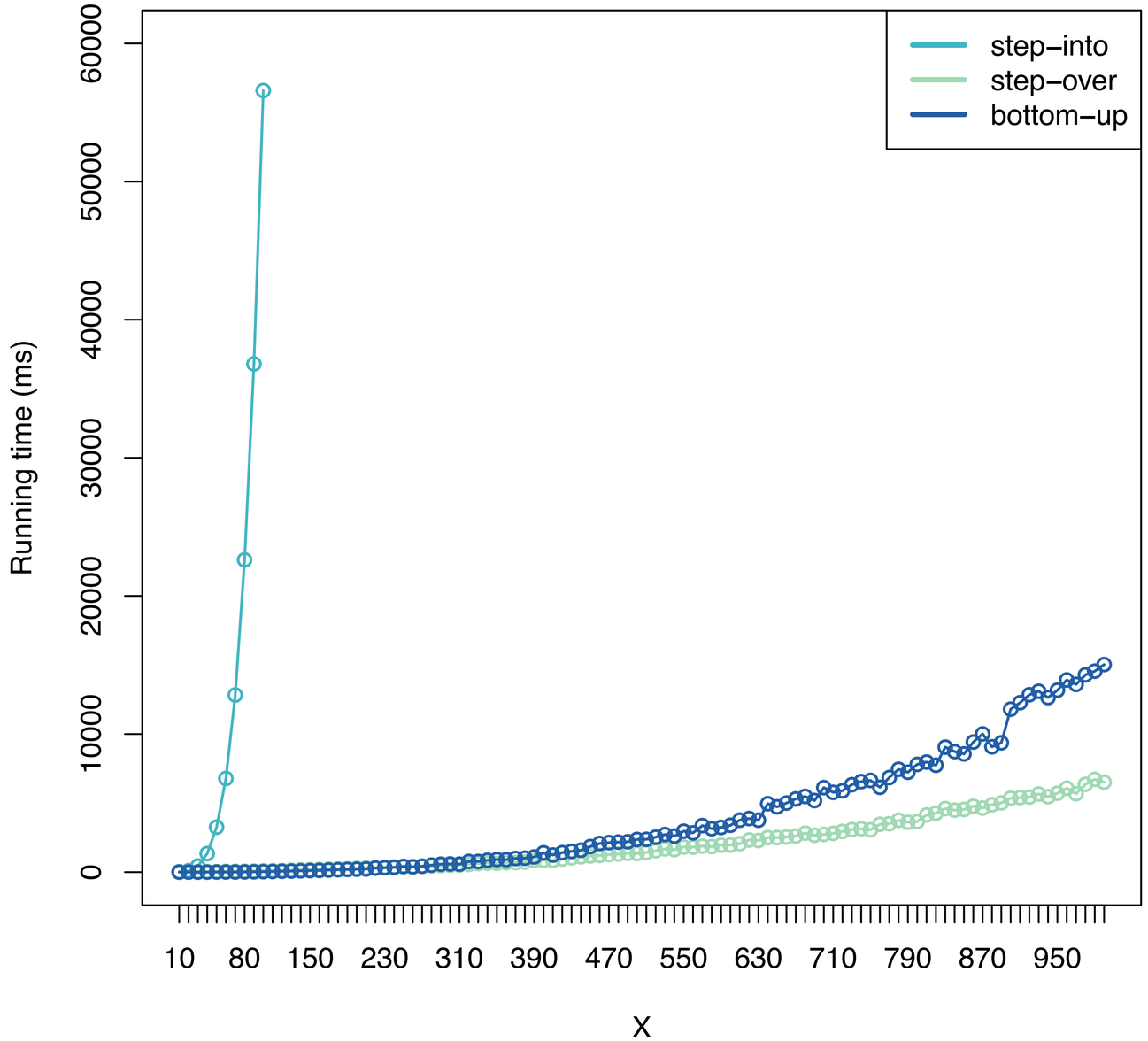
Performance problem

Simulate step-over and resume through repeated step-into?

Evaluated on synthetic cyclic graphs, computing path relation



Running time for different graph sizes

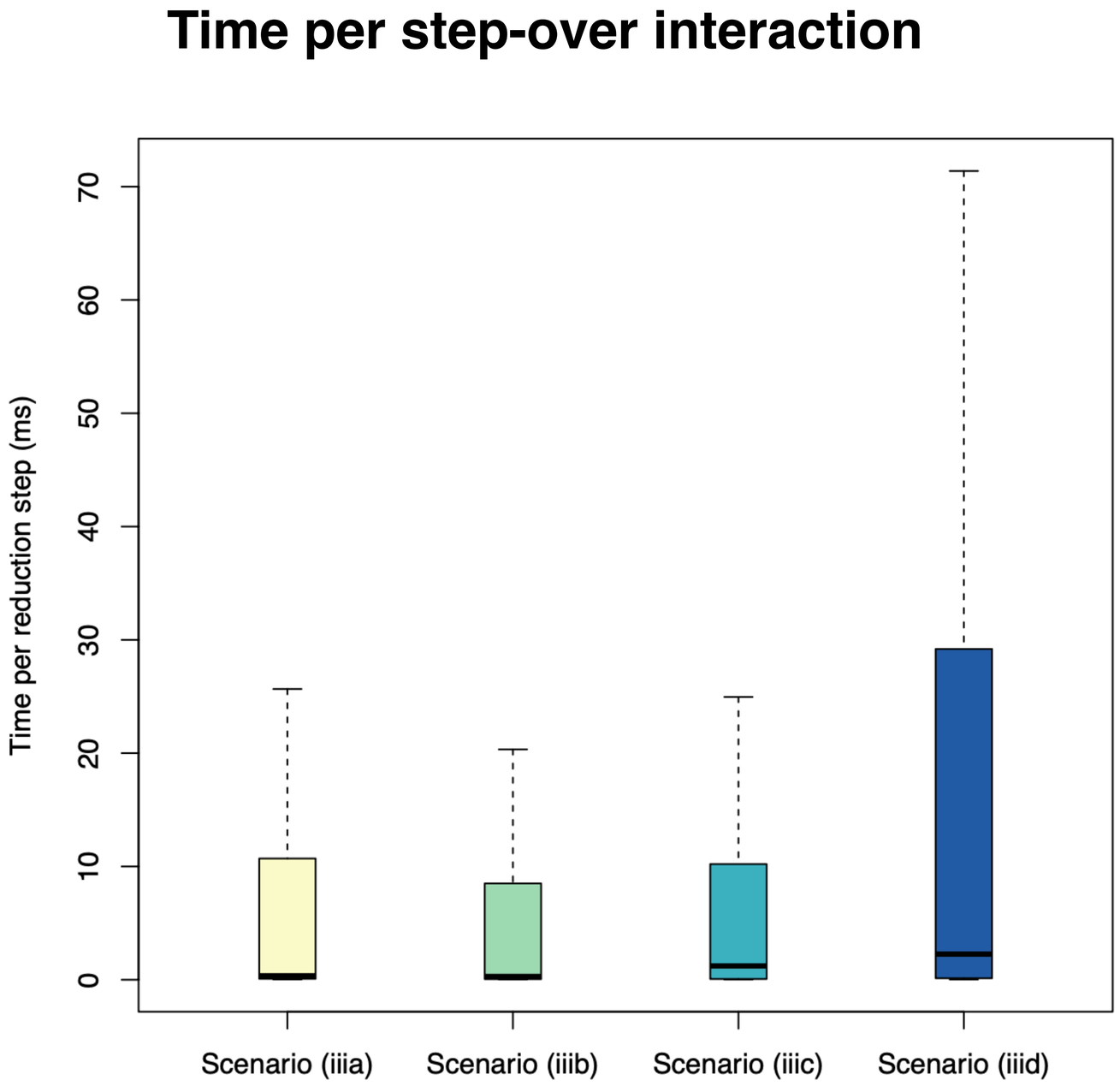


Step-over performance

Implemented in IncA

Evaluated on Doop varPointsTo

Step over, except:
a) VarPointsTo
b) VarPointsTo, StaticFieldPointsTo
c) VarPointsTo, InstanceFieldPointsTo
d) VarPointsTo, Reachable



Interactive Debugging of Datalog

Top-down semantics: program exploration
Hybrid semantics: efficient step-over/resume

Vision: Datalog as IR with rich tooling