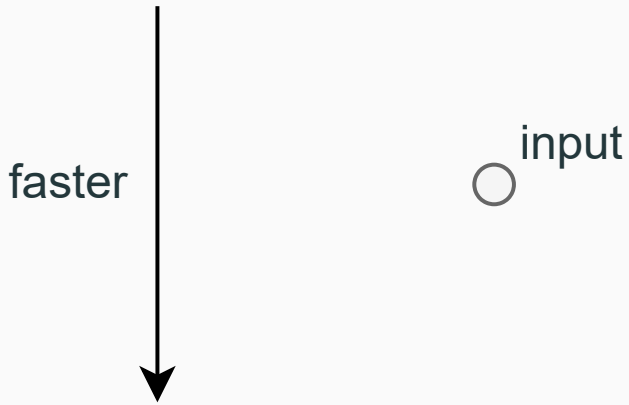
Soufflé is Baked Egg

Building compiler optimizations in Datalog

Remy Wang, Yihong Zhang, Max Willsey, Philip Zucker, Zachary Tatlock

April 2022 @ HYTRADBOI

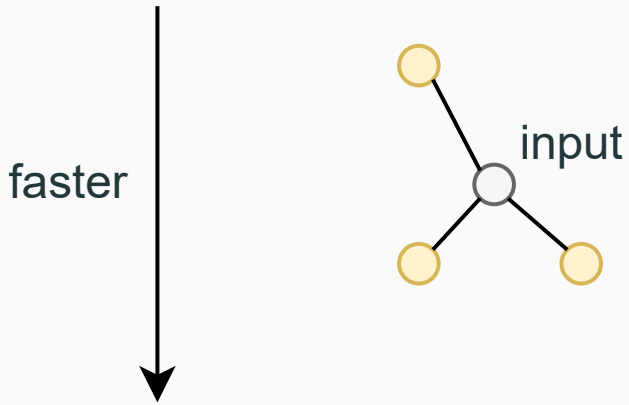
How does a compiler optimize?



Cascade style[1]: explore the space of equivalent programs.

[1] A.k.a. equality saturation.

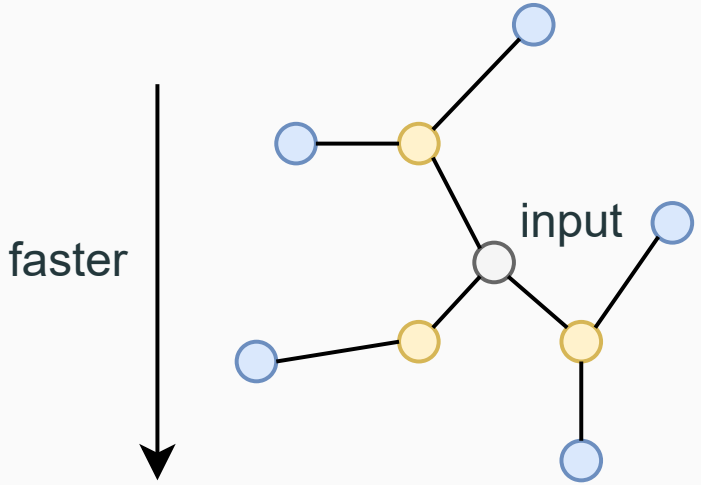
How does a compiler optimize?



Cascade style[1]: explore the space of equivalent programs.

[1] A.k.a. equality saturation.

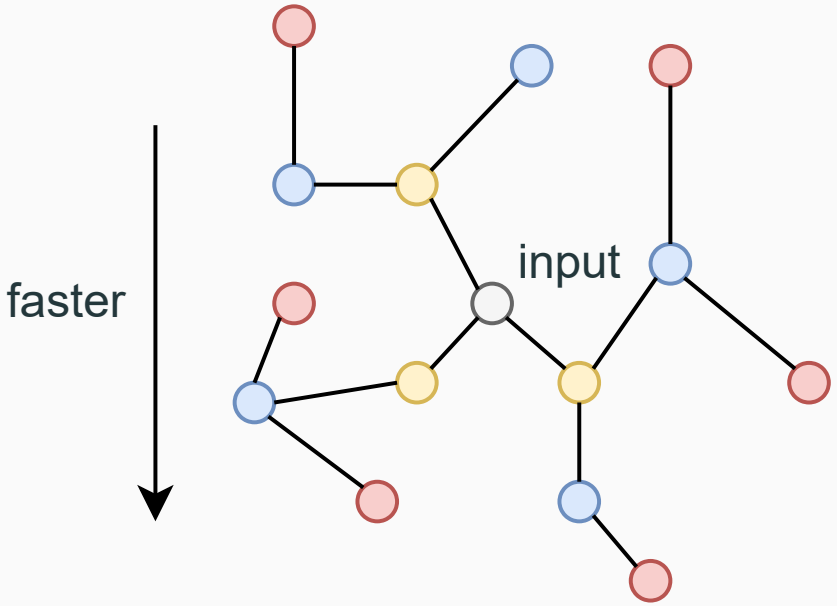
How does a compiler optimize?



Cascade style[1]: explore the space of equivalent programs.

[1] A.k.a. equality saturation.

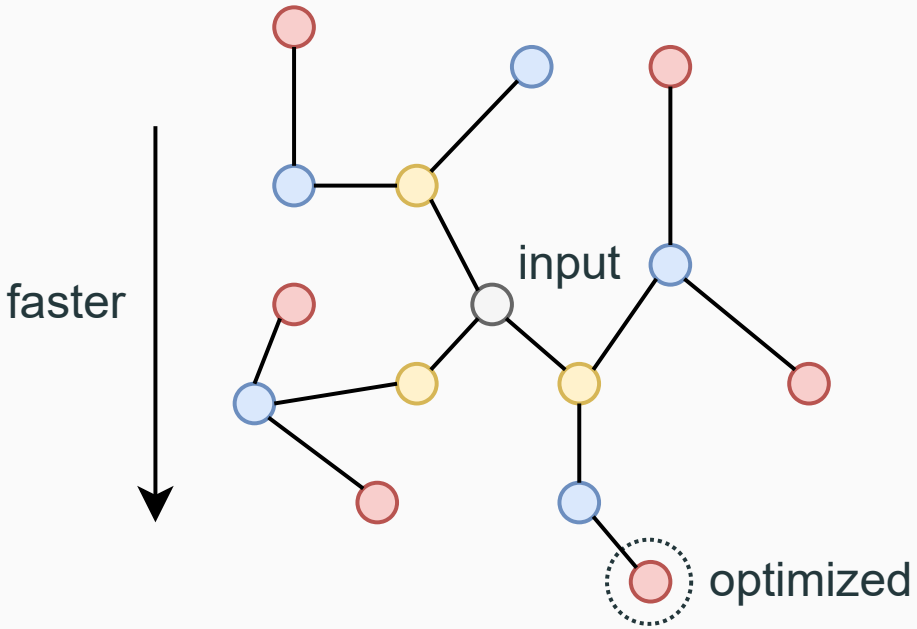
How does a compiler optimize?



Cascade style[1]: explore the space of equivalent programs.

[1] A.k.a. equality saturation.

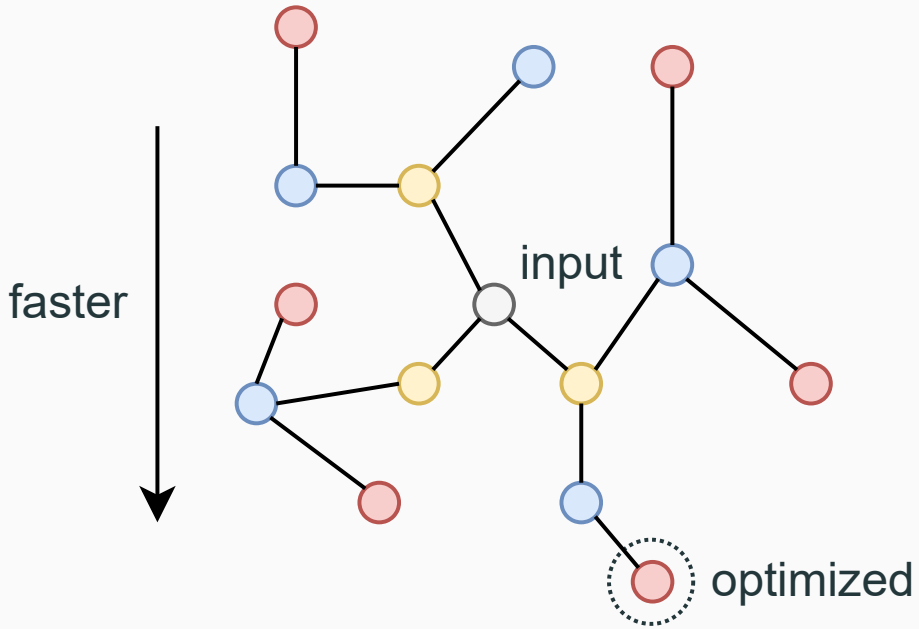
How does a compiler optimize?



Cascade style[1]: explore the space of equivalent programs.

[1] A.k.a. equality saturation.

How does a compiler optimize?



Cascade style[1]: explore the space of equivalent programs.

Main loop: match rewrite rules on programs to grow the set.

[1] A.k.a. equality saturation.

How does a compiler optimize?

Main **loop**: **match** rewrite **rules** on programs to **grow the set**.

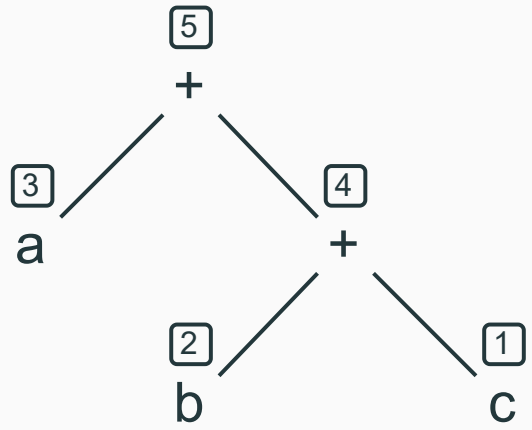
How does a compiler optimize?

Main **loop**: **match** rewrite **rules** on programs to **grow the set**.

Datalog does that all day, every day!

Optimization in Datalog: Encoding Terms

"Flat" representation of terms:



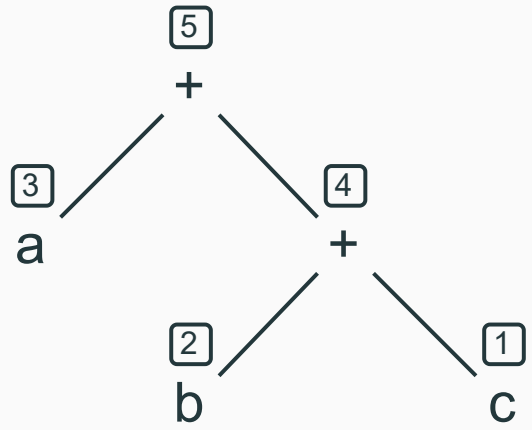
$\Rightarrow$

var	id_{var}
c	1
b	2
a	3

arg_1	arg_2	id_+
2	1	4
3	2	5

Optimization in Datalog: Encoding Terms

"Flat" representation of terms:



$\Rightarrow$

var	id_{var}
c	1
b	2
a	3

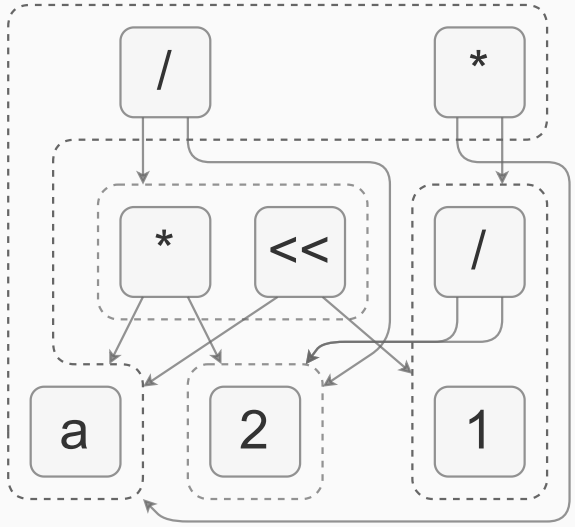
arg_1	arg_2	id_+
2	1	4
3	2	5

In Soufflé:

```
.type Exp = Add{x:Exp,y:Exp} | Var{n:symbol}
.decl add(x:Exp,y:Exp,id:Exp)
.decl var(v:symbol,id:Exp)
add(x,y,id) :- id=$Add(x,y).
```

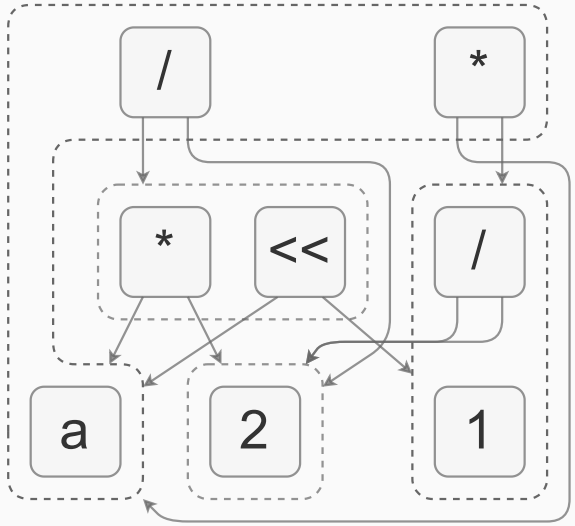
Optimization in Datalog: Encoding Terms

Group equivalent subexpressions into **equivalent classes**:



Optimization in Datalog: Encoding Terms

Group equivalent subexpressions into **equivalent classes**:




```
.decl eql(x:Exp,y:Exp) eqrel
// keep only canonical Exp
add(x,y,z) <= add(a,b,c) :-
  eql(x,a), eql(y,b), eql(z,c),
  a<=x, b<=y, c<=z.
```

Optimization in Datalog: Encoding Rewrites

A rewrite can add new terms and new equalities:

```
// comm-add
eql(xy,yx), add(x,y,xy) :-
  add(y,x,yx), xy=$Add(x,y).
```

Optimization in Datalog: Encoding Rewrites

A rewrite can add new terms and new equalities:

```
// comm-add
eql(xy,yx), add(x,y,xy) :-
  add(y,x,yx), xy=$Add(x,y).

// assoc-add
eql(xy_z,x_yz), add(xy,z,xy_z), add(x,y,xy) :-
  add(y,z,yz), add(x,yz,x_yz),
  xy = $Add(x,y), xy_z = $Add(xy,z).
```

Optimization in Datalog: What's the Point?

Performance:

- Join algorithms for e-matching [ZWWT22].
- Semi-naïve/IVM (TreeToaster [BNKZ21])

Optimization in Datalog: What's the Point?

Performance:

- Join algorithms for e-matching [ZWWT22].
- Semi-naïve/IVM (TreeToaster [BNKZ21])

Expressiveness:

- Composable lattice-style program analyses (Flix [MYL16])
- Leverage existing analysis (DOOP [BS09])

Optimization in Datalog: What's Left?

Performance:

- Actually efficient equivalence relation (`eqrel` in Soufflé [NZSS19]).
- Congruence algorithms [WNW$^+$21] (semi-naïve?).

Optimization in Datalog: What's Left?

Performance:

- Actually efficient equivalence relation (`eqrel` in Soufflé [NZSS19]).
- Congruence algorithms [WNW+21] (semi-naïve?).

Expressiveness: what to do with this new-found power??

- Improve precision of analyses [Ste96]?
- Composing transformation & analyses[LGC02]?
- Decompiler [GBSS19]?
- Relations other than equivalence (e.g. implication)?
- ...

References i

Darshana Balakrishnan, Carl Nuessle, Oliver Kennedy, and Lukasz Ziarek, *Treetoaster: Towards an ivm-optimized compiler*, SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021 (Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, eds.), ACM, 2021, pp. 155–167.

Martin Bravenboer and Yannis Smaragdakis, *Strictly declarative specification of sophisticated points-to analyses*, Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando,

References ii

Florida, USA (Shail Arora and Gary T. Leavens, eds.), ACM, 2009, pp. 243–262.

Neville Grech, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis, *Gigahorse: thorough, declarative decompilation of smart contracts*, Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019 (Joanne M. Atlee, Tevfik Bultan, and Jon Whittle, eds.), IEEE / ACM, 2019, pp. 1176–1186.

References iii

Sorin Lerner, David Grove, and Craig Chambers, *Composing dataflow analyses and transformations*, Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002 (John Launchbury and John C. Mitchell, eds.), ACM, 2002, pp. 270–282.

Magnus Madsen, Ming-Ho Yee, and Ondrej Lhoták, *From datalog to flix: a declarative language for fixed points on lattices*, Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016 (Chandra Krintz and Emery D. Berger, eds.), ACM, 2016, pp. 194–208.

References iv


Patrick Nappa, David Zhao, Pavle Subotic, and Bernhard Scholz, *Fast parallel equivalence relations in a datalog compiler*, 28th International Conference on Parallel Architectures and Compilation Techniques, PACT 2019, Seattle, WA, USA, September 23-26, 2019, IEEE, 2019, pp. 82–96.


Bjarne Steensgaard, *Points-to analysis in almost linear time*, Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996

References v

(Hans-Juergen Boehm and Guy L. Steele Jr., eds.), ACM Press, 1996, pp. 32–41.


Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha, *egg: Fast and extensible equality saturation*, Proc. ACM Program. Lang. **5** (2021), no. POPL, 1–29.


Yihong Zhang, Yisu Remy Wang, Max Willsey, and Zachary Tatlock, *Relational e-matching*, Proc. ACM Program. Lang. **6** (2022), no. POPL, 1–22.