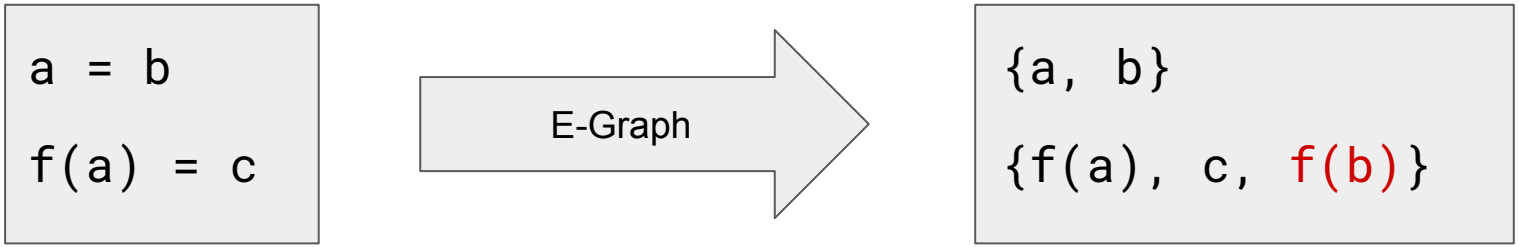
Slotted E-Graphs

E-Graphs with built-in Name Binding

Rudi Schneider, Thomas Kœhler, Michel Steuwer

E-Graphs

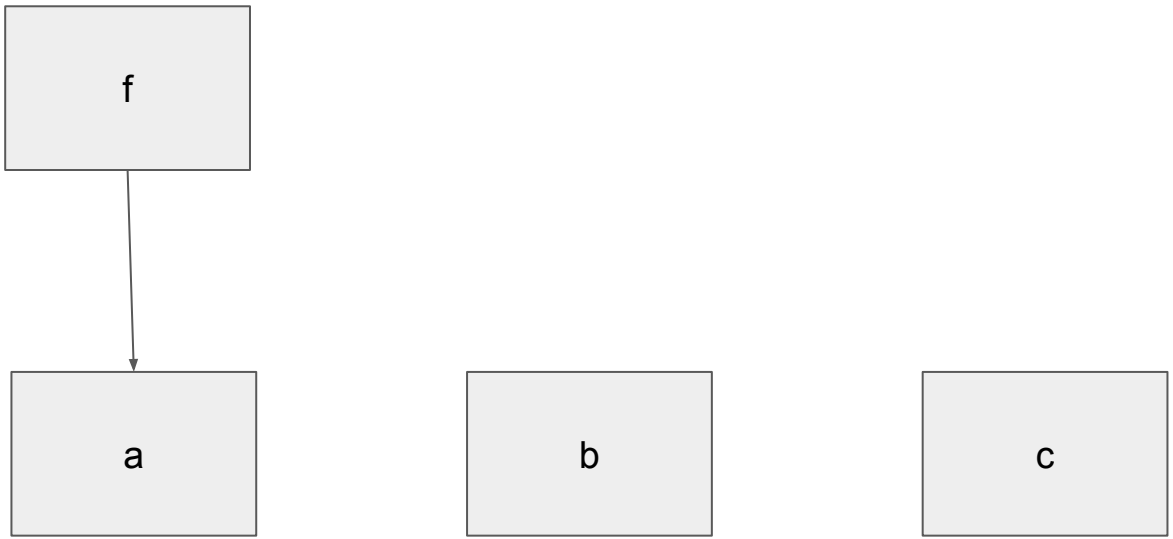
- **E**quivalence-**Graph**
- represents equivalence classes of terms (congruence closure)



- useful for equality of uninterpreted functions in SMT solvers (eg. Z3)
- compiler optimization (i.e. find equivalent but *better* term)

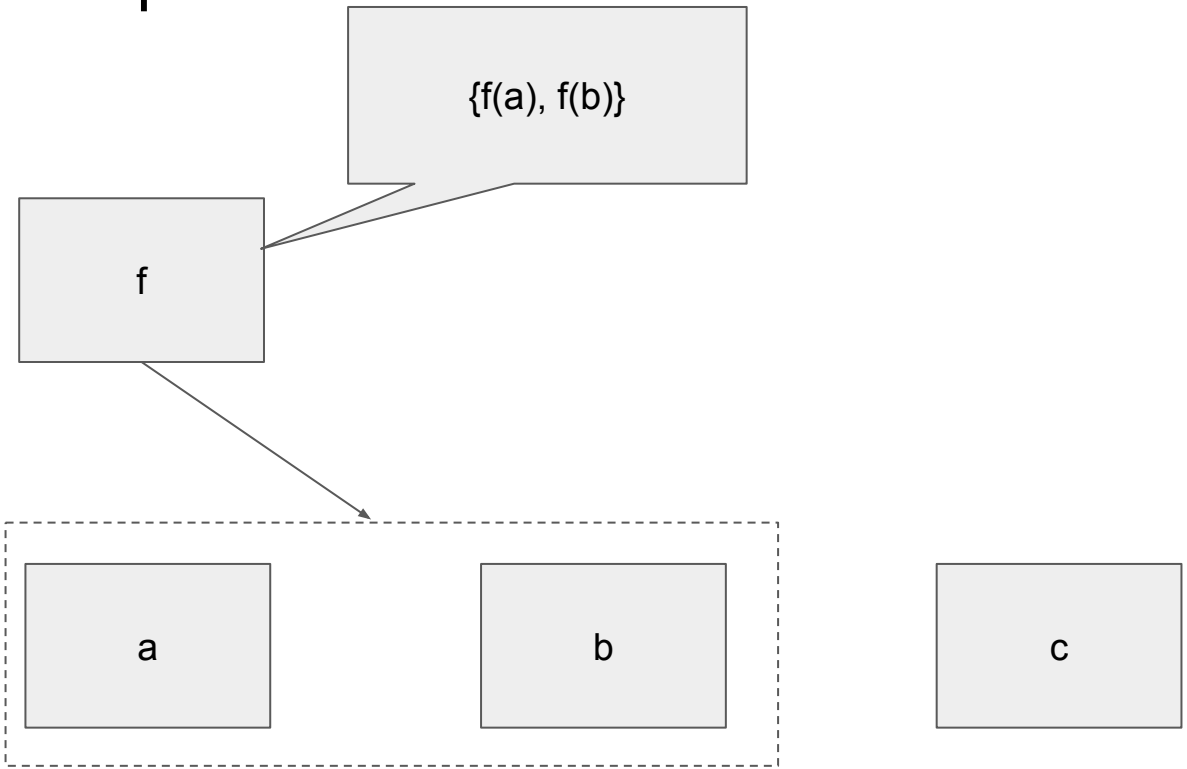
E-Graphs

$a=b \wedge f(a)=c$



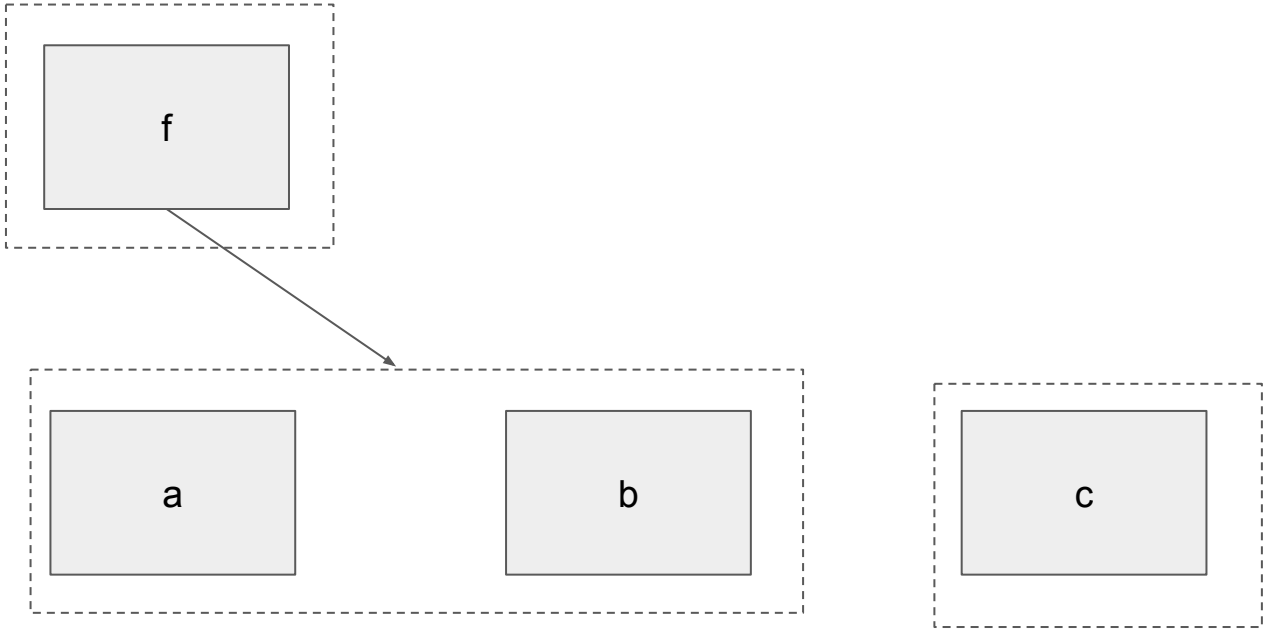
E-Graphs

$a=b \wedge f(a)=c$



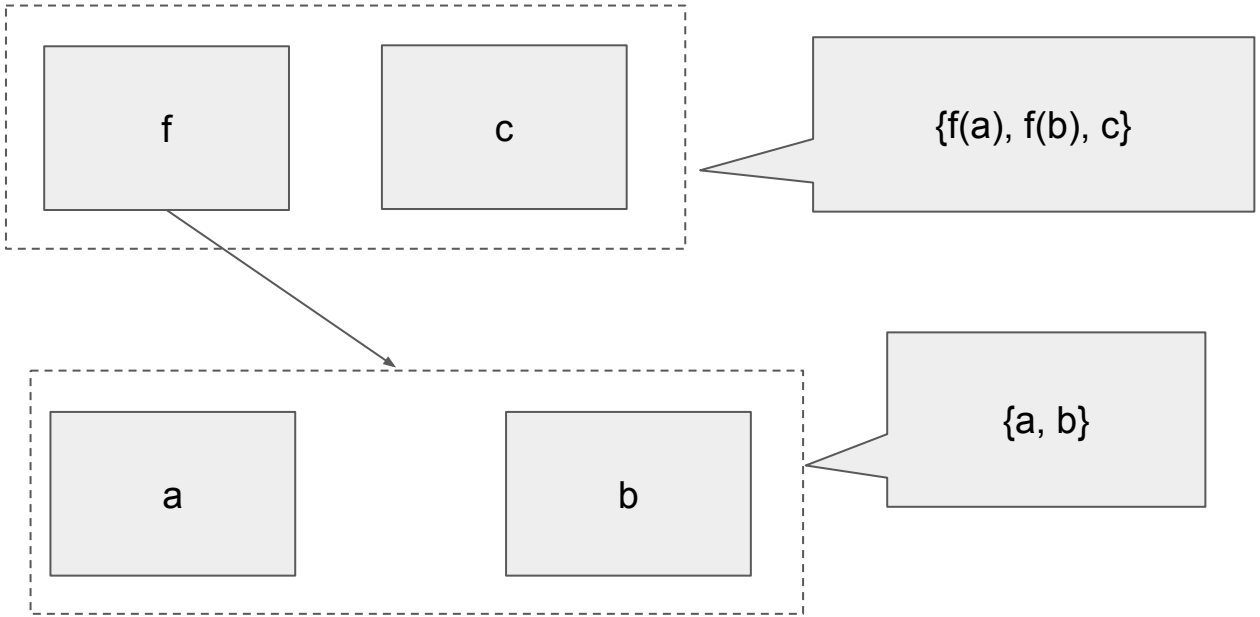
E-Graphs

$a=b \wedge f(a)=c$



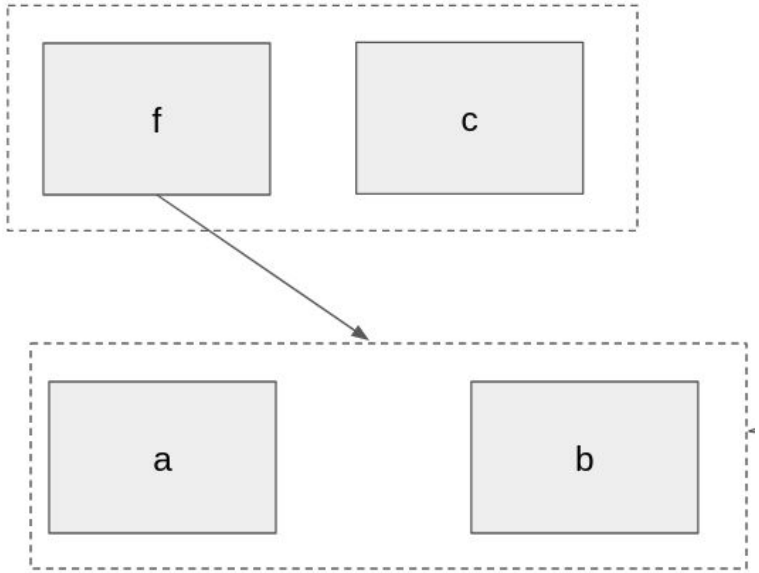
E-Graphs

$a=b \wedge f(a)=c$



E-Graphs - Terminology

```
term ::= c | f(term...)

e-node ::= c | f(id...)

e-class ::= {e-node, ...}
```



E-Graphs - Data Structure

`classes :: Map id e-class`

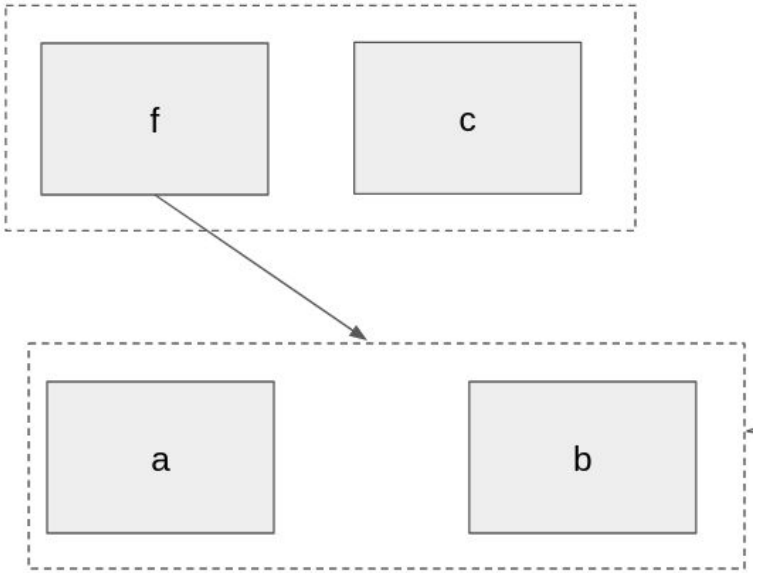
- maps an E-Class id to their e-node set

`unionfind :: Map id id`

- `unionfind[x]` is the “representative” from the equivalence class of `x`

`hashcons :: Map e-node id`

- maps an e-node to the e-class containing it
- allows fast lookup of e-nodes and terms



Equality Saturation

Equality Saturation

We want to simplify `(a * 2) / 2` using a set of rewrite rules.

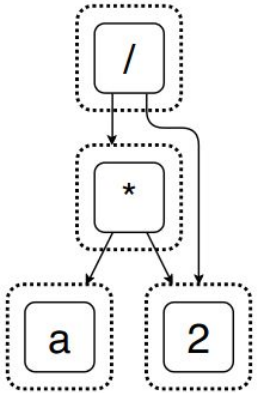
useful

```
(?x * ?y) / ?z → ?x * (?y / ?z)
?x / ?x → 1
?x*1 → ?x
```

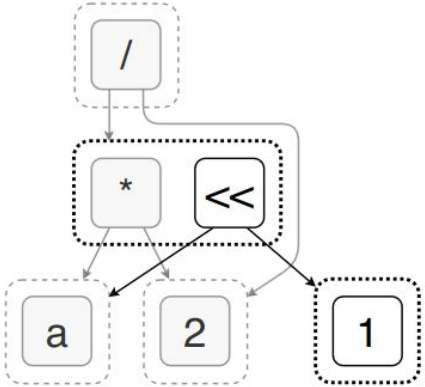
not so useful

```
?x * 2 → ?x << 1
?x * ?y → ?y * ?x
?x → ?x*1
```

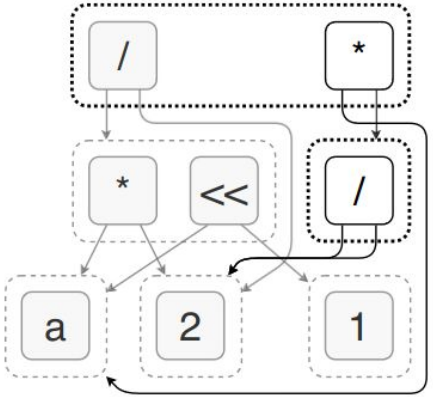
Equality Saturation



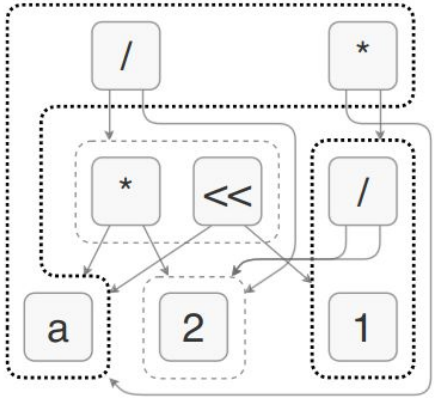
(a) Initial e-graph contains $(a \times 2)/2$.



(b) After applying rewrite $x \times 2 \rightarrow x \ll 1$.

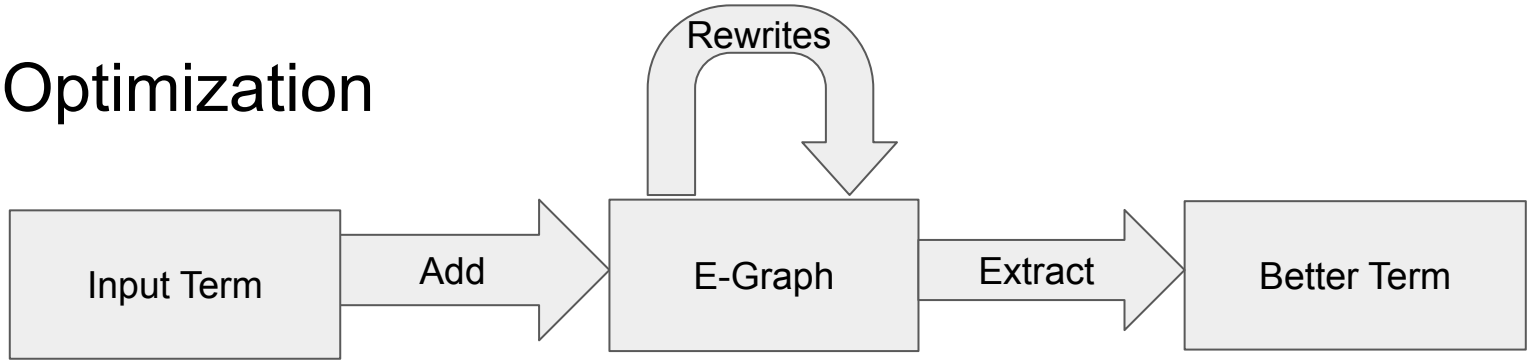


(c) After applying rewrite $(x \times y)/z \rightarrow x \times (y/z)$.

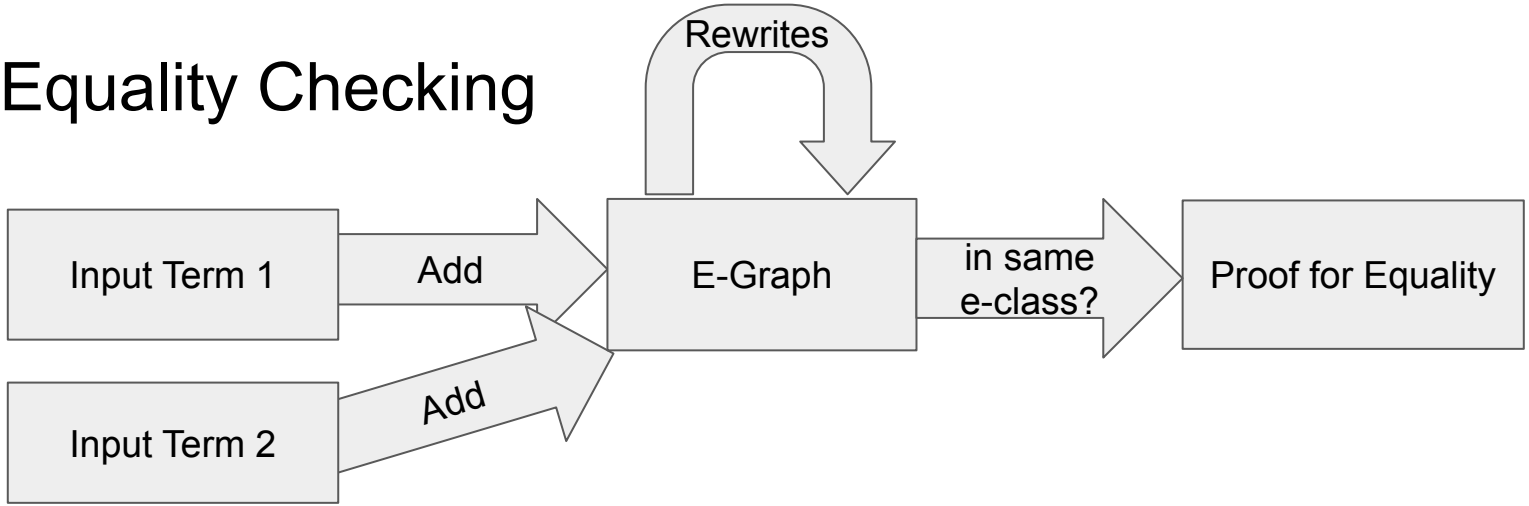


(d) After applying rewrites $x/x \rightarrow 1$ and $1 \times x \rightarrow x$.

Optimization



Equality Checking



Name Binding

Name Binding

$$\lambda x.\ x + 1$$

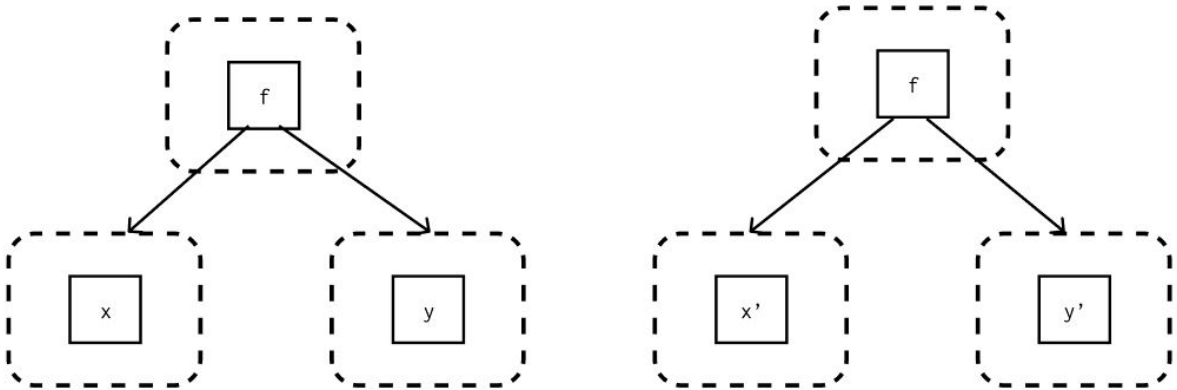
$$\sum_{i=1}^{N} A_i + B_i$$

$$\forall x.\ x + 1 > x$$

```
for x in range(N):
    foo(x)
```

E-Graphs are bad at Name Binding

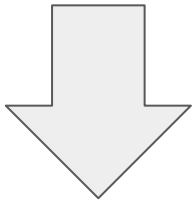
- Consider the terms `f(x,y)` and `f(x',y')`
- Perfect use-case for sharing!



- Renaming variables breaks sharing!

Why don’t just use DeBruijn Indices?

```
λ. (λ. f(%1) %0) arg
```



beta-reduction

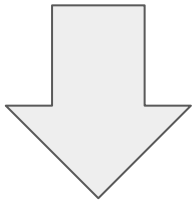
```
λ. f(%0) arg
```

- Again, no sharing between `f(%0)` and `f(%1)`

- Adding/removing binders also breaks sharing!

Why don’t just use DeBruijn Indices?

```
λ. (λ. f(%1) %0) arg
```



beta-reduction

```
λ. f(%0) arg
```

- Again, no sharing between `f(%0)` and `f(%1)`

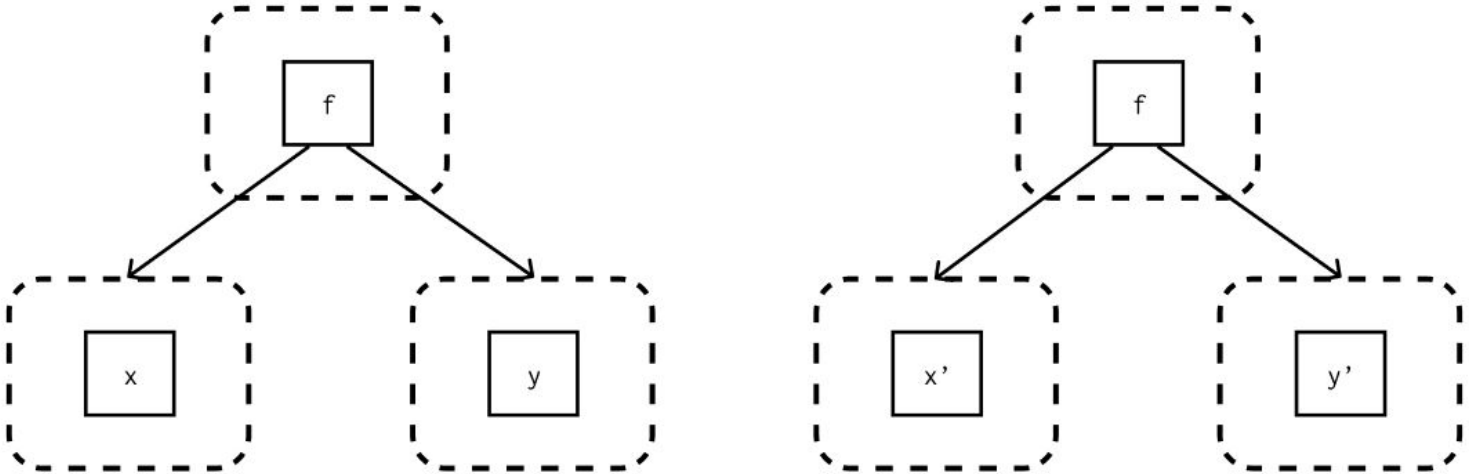
- Adding/removing binders also breaks sharing!

Slotted E-Graphs

Slotted E-Graphs

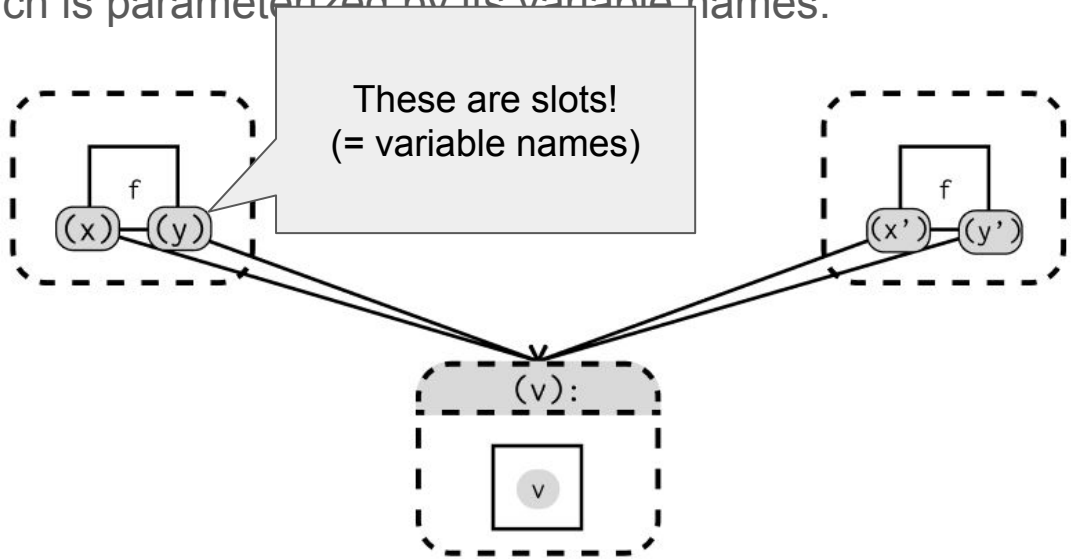
Key idea: Unify e-nodes that are equivalent up to renaming of variables,

into an e-class which is parameterized by its variable names.



Slotted E-Graphs

Key idea: Unify e-nodes that are equivalent up to renaming of variables, into an e-class which is parameterized by its variable names.

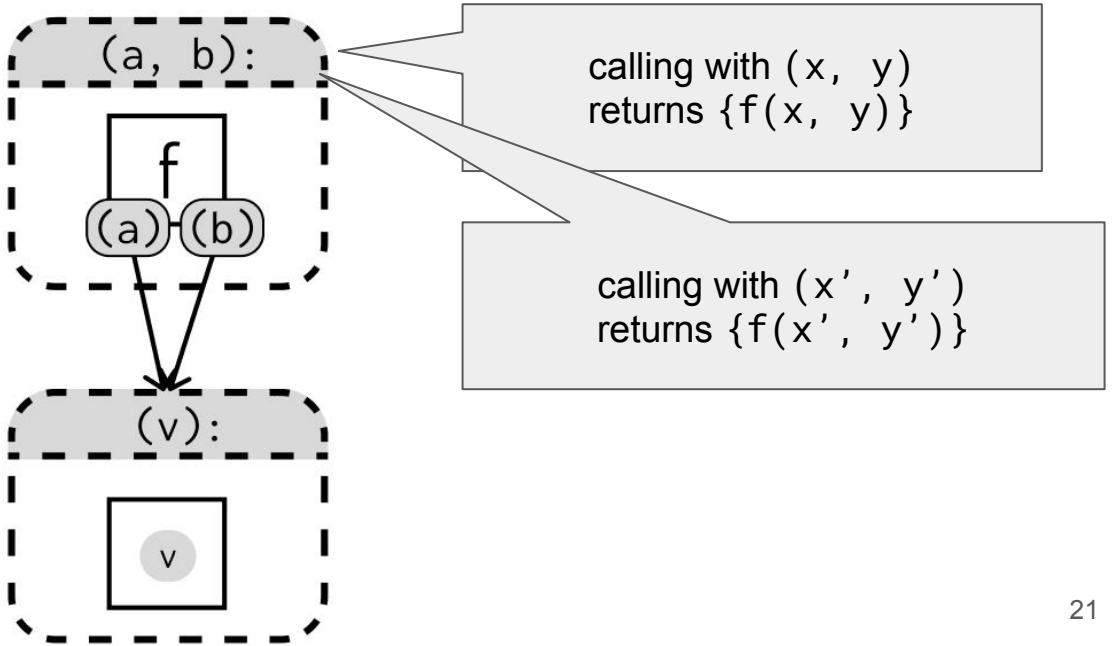


Slotted E-Graphs

Key idea: Unify e-nodes that are equivalent up to renaming of variables,

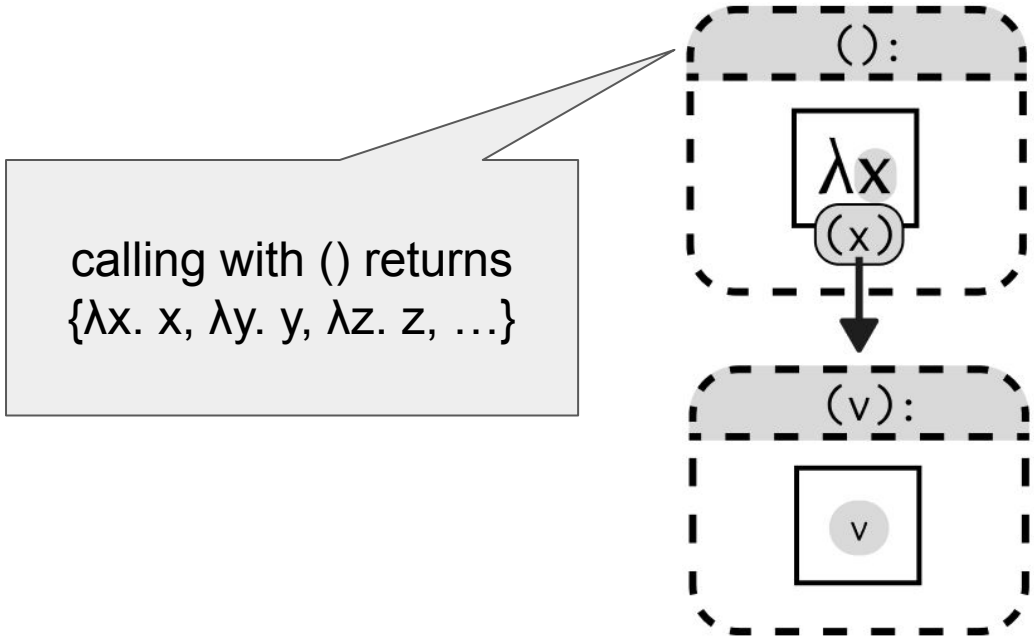
into an e-class which is parameterized by its variable names.

- Conventional E-Class: Set of equivalent terms
- Slotted E-Class: Function from variable names to set of equivalent terms



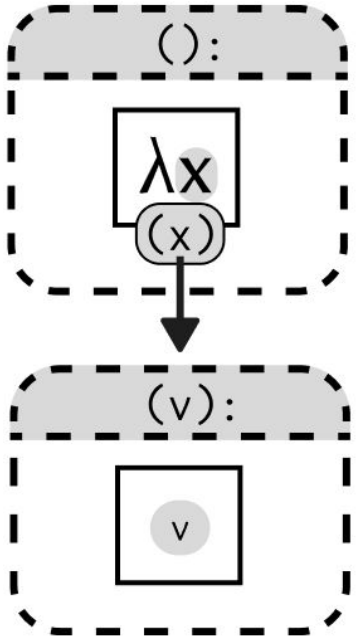
Slotted E-Graphs - Binders

- Example: λx. x



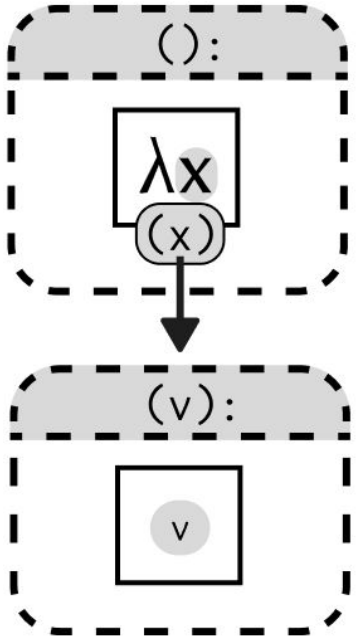
Slotted E-Graphs

```
term ::= c | f(term...)

e-node ::= c | f(id...)

e-class ::= {e-node, ...}
```



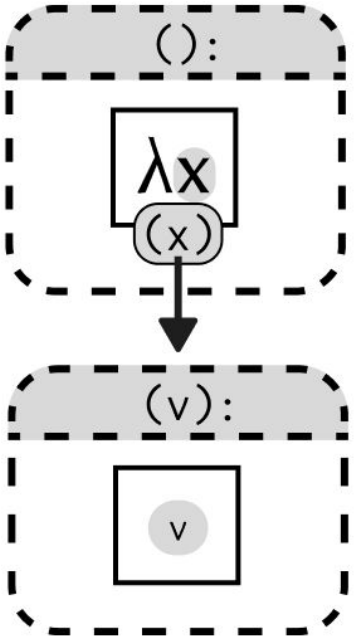
Slotted E-Graphs

```
term ::= c | f(term...) | λslot. term | slot

e-node ::= c | f(id...)

e-class ::= {e-node, ...}
```



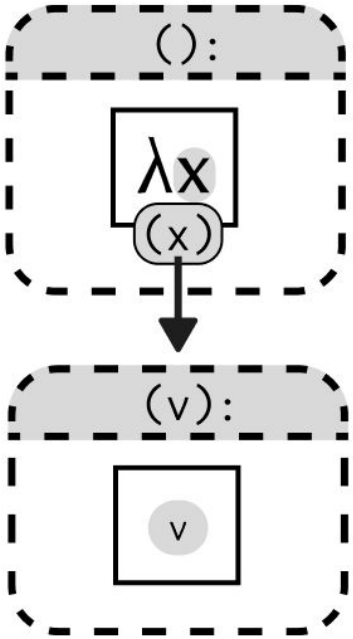
Slotted E-Graphs

```
term ::= c | f(term...) | λslot. term | slot

e-node ::= c | f(id...) | λslot. id | slot

e-class ::= {e-node, ...}
```



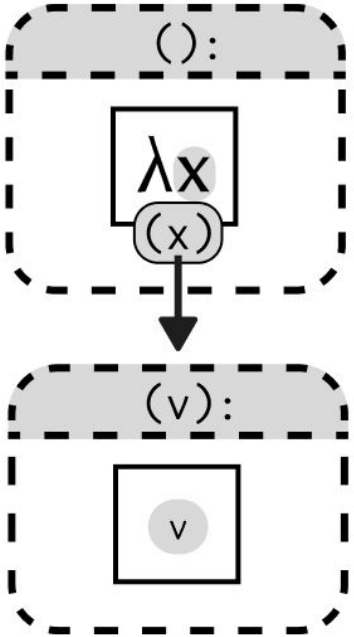
Slotted E-Graphs

```
term ::= c | f(term...) | λslot. term | slot

e-node ::= c | f(id...) | λslot. id | slot

e-class ::= {e-node, ...}

invocation ::= id[slot...]
```



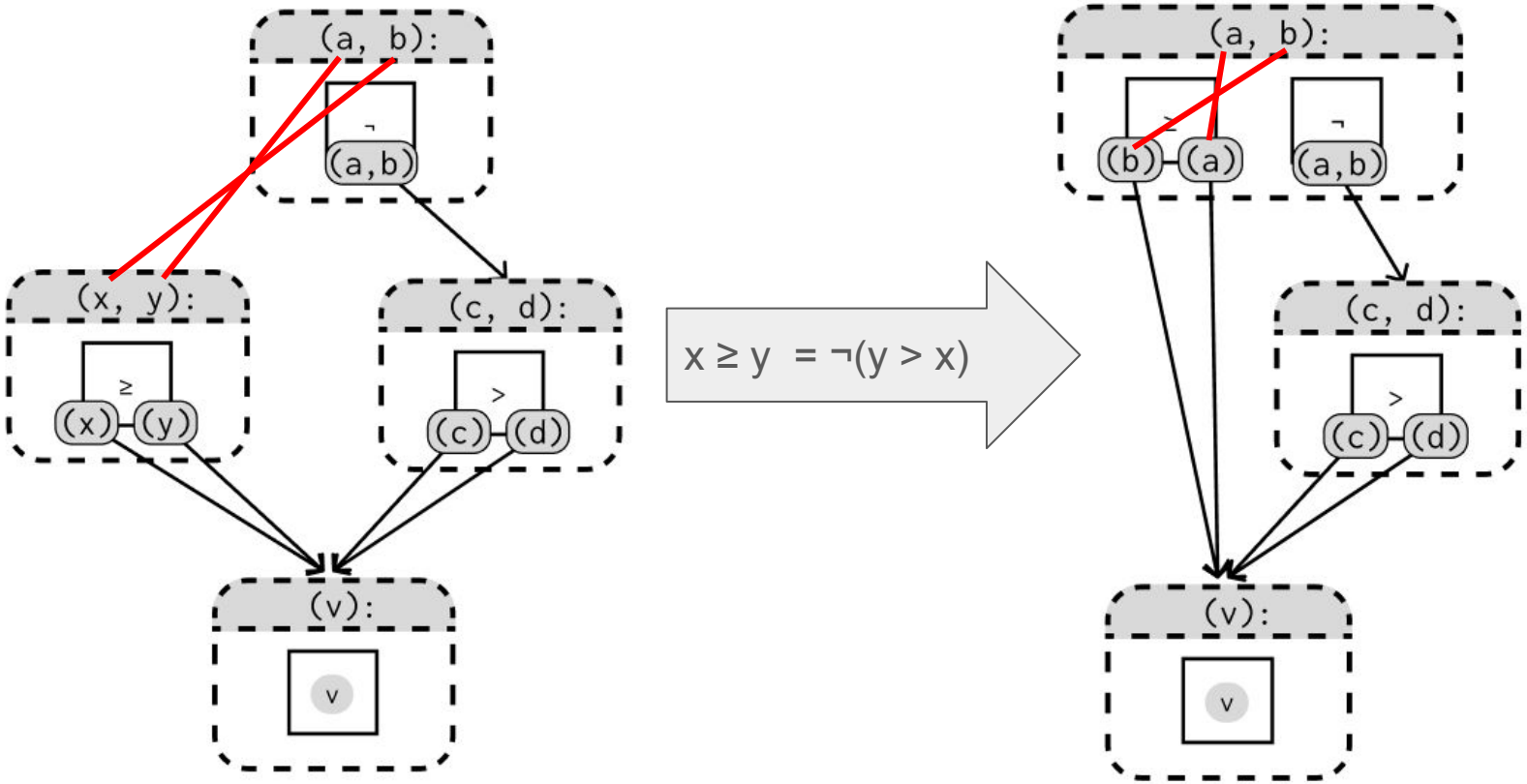
Slotted E-Graphs

```
term ::= c | f(term...) | λslot. term | slot

e-node ::= c | f(invocation...)

             | λslot. invocation | slot

e-class ::= {e-node, ...}

invocation ::= id[slot...]
```

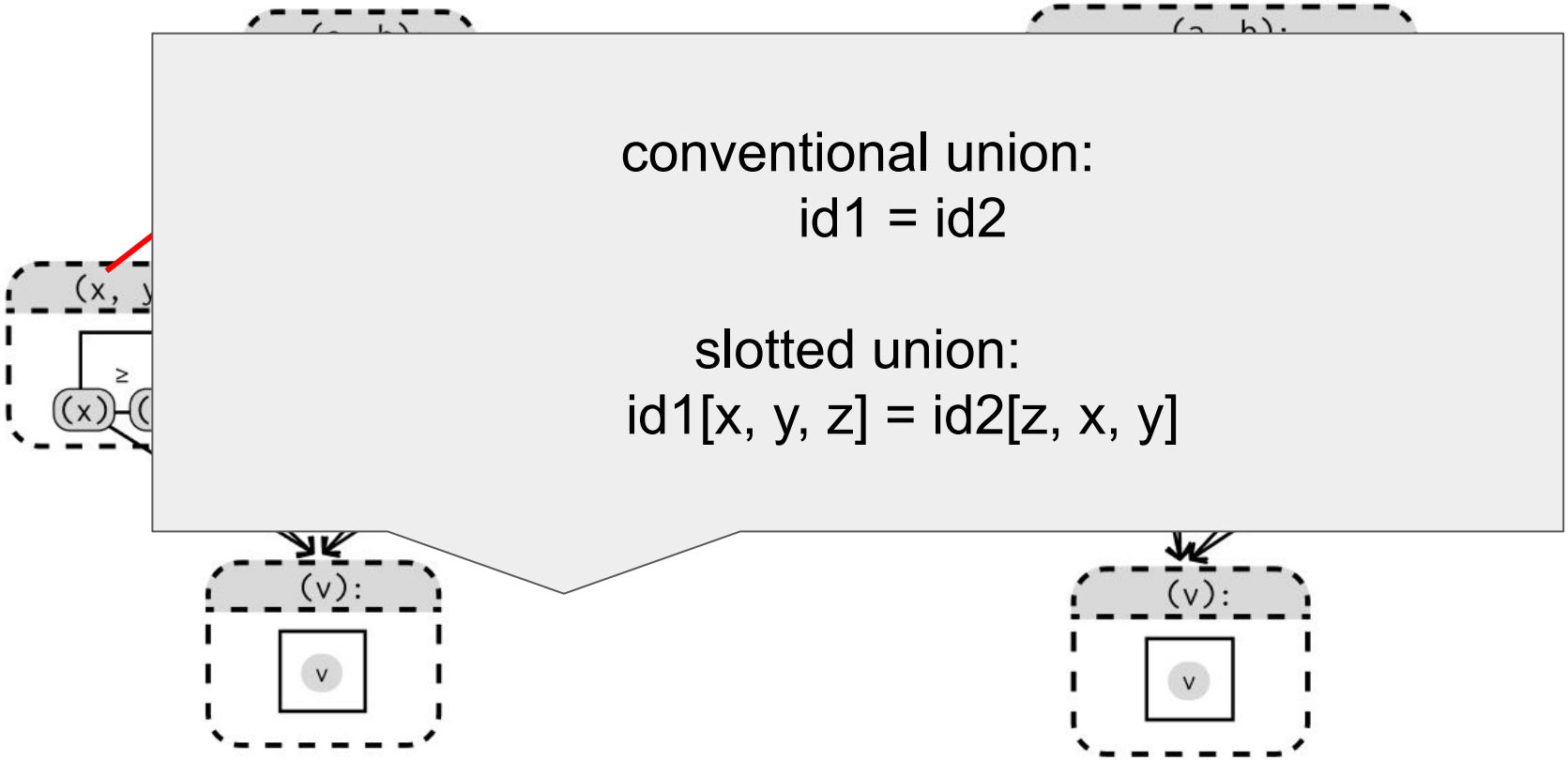


Challenges

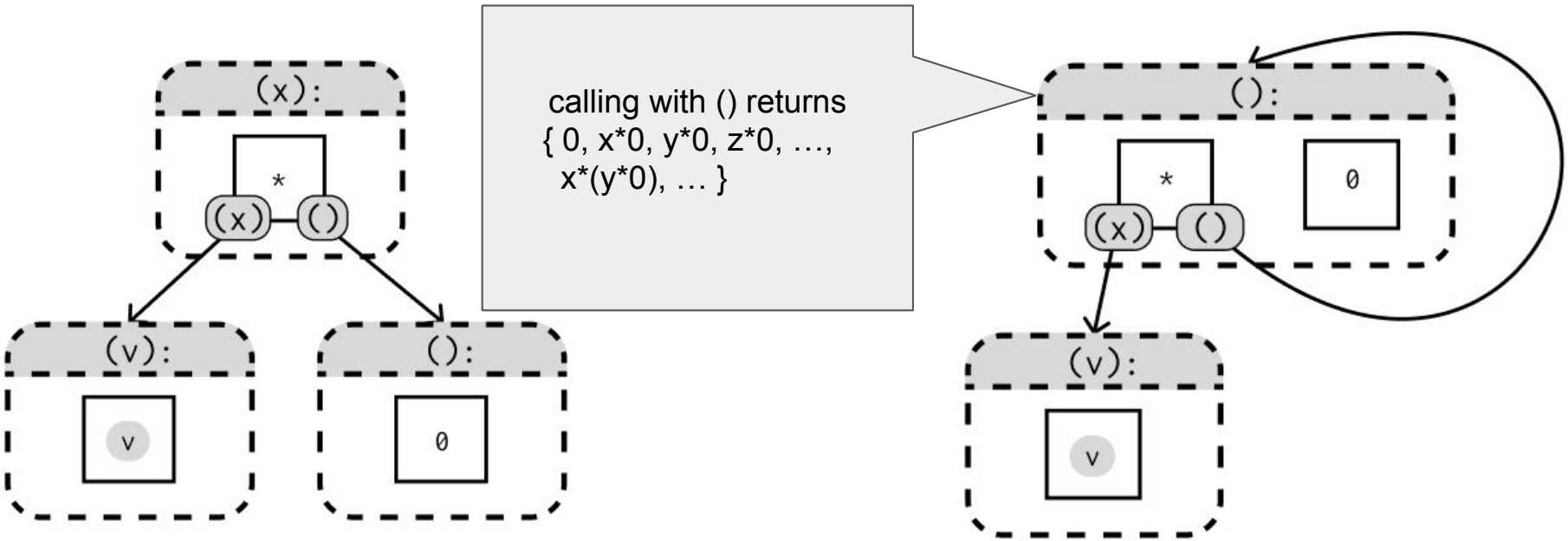
Union just got complicated!



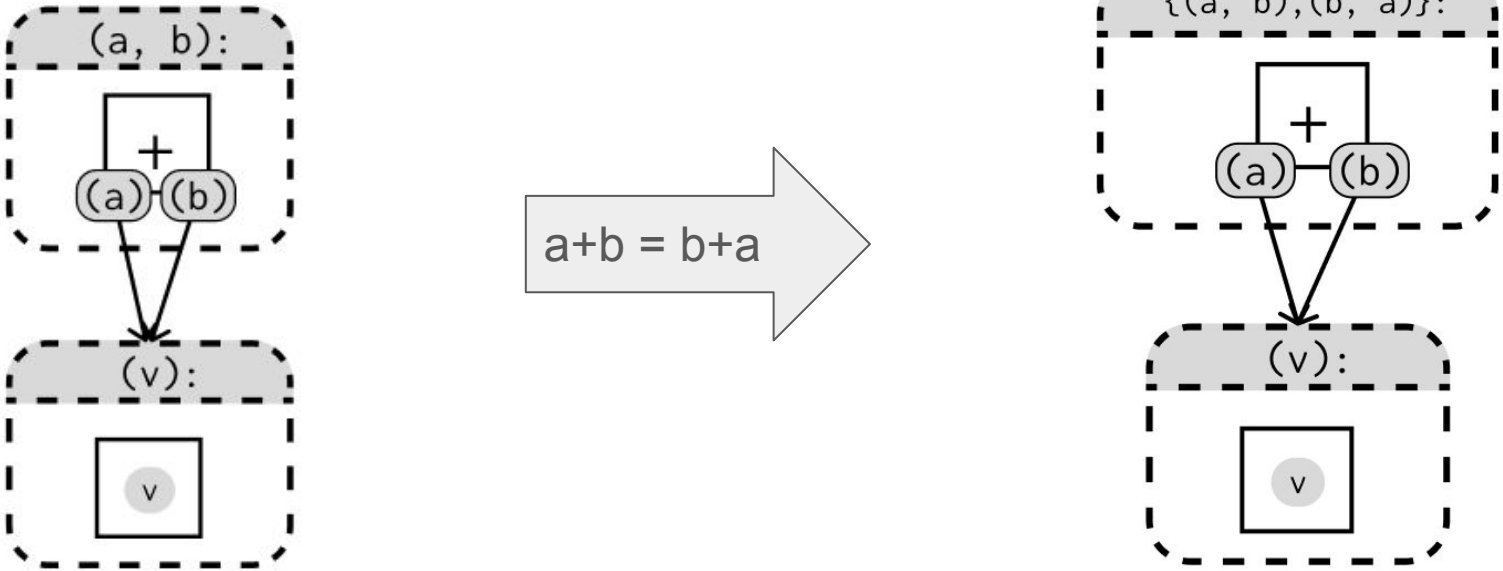
Union just got complicated!



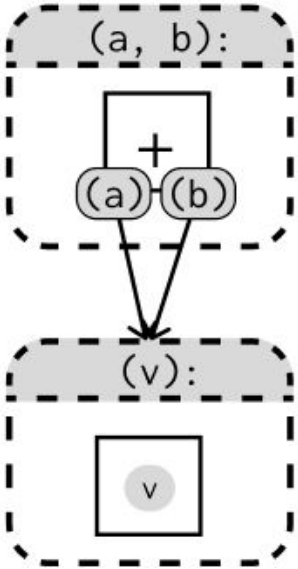
Challenge I: Parameter Mismatch



Challenge II: Symmetry

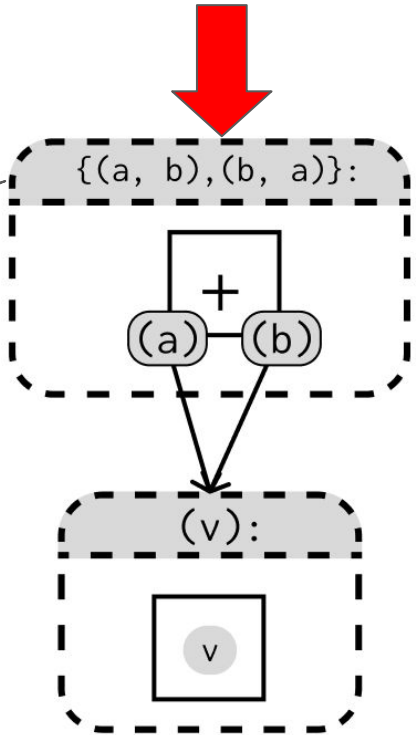


Challenge II: Symmetry



calling with (x, y)
returns {x+y, y+x}

Permutation Group



Data Structure

```
classes :: Map id e-class

unionfind :: Map id id

hashcons :: Map e-node id
```

conventional unionfind:
id1 → id2

slotted unionfind:
id1[x, y, z] → id2[z, x, y]

modulo slot names:
id1[0, 1, 2] → id2[2, 0, 1]

Data Structure

```
classes :: Map id e-class

unionfind :: Map id id

hashcons :: Map e-node id
```

conventional hashcons:
f(id1, id2) → id3

slotted hashcons:
f(id1[x, y], id2[z, y]) → id3[y, z, x]

modulo slot names:
f(id1[0, 1], id2[2, 1]) → id3[1, 2, 0]

Data Structure

```
classes :: Map id e-class

unionfind :: Map invocation invocation

hashcons :: Map e-node invocation
```

Data Structure

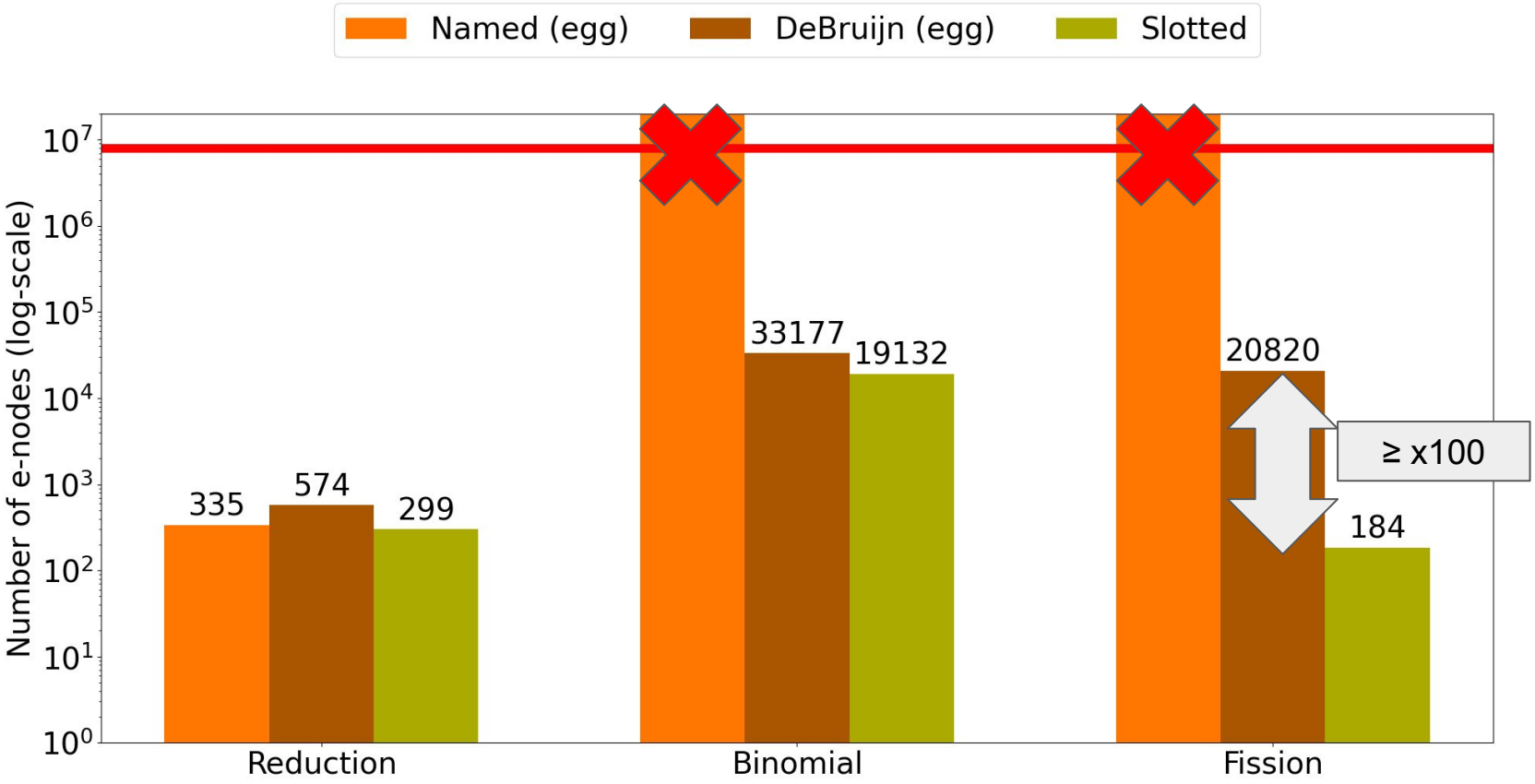
```
classes :: Map id e-class

unionfind :: Map invocation invocation

hashcons :: Map e-node invocation

symmetries :: Map id perm-group
```

Evaluation

Evaluation

- Task: Optimize RISE term using equality saturation
- RISE is a functional data-parallel language [ICFP 2020]
- We compared against the **Named** and **De-Bruijn** implementations from Thomas Kœhlers PhD thesis

Evaluation


Named (egg)
DeBruijn (egg)
Slotted
Number of e-nodes (log-scale)
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
335
574
299
33177
19132
20820
184
≥ x100
Reduction
Binomial
Fission

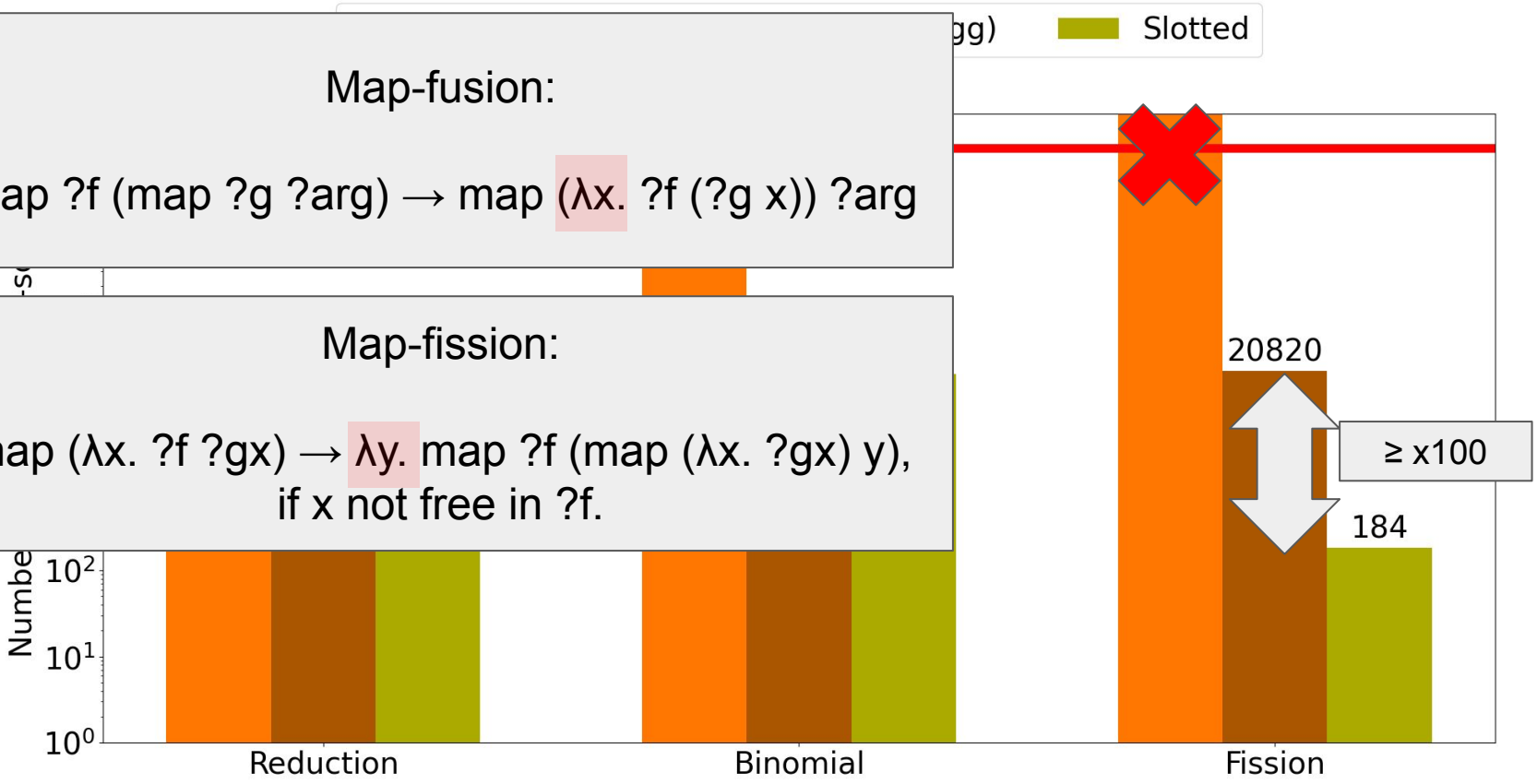
Evaluation

Map-fusion:

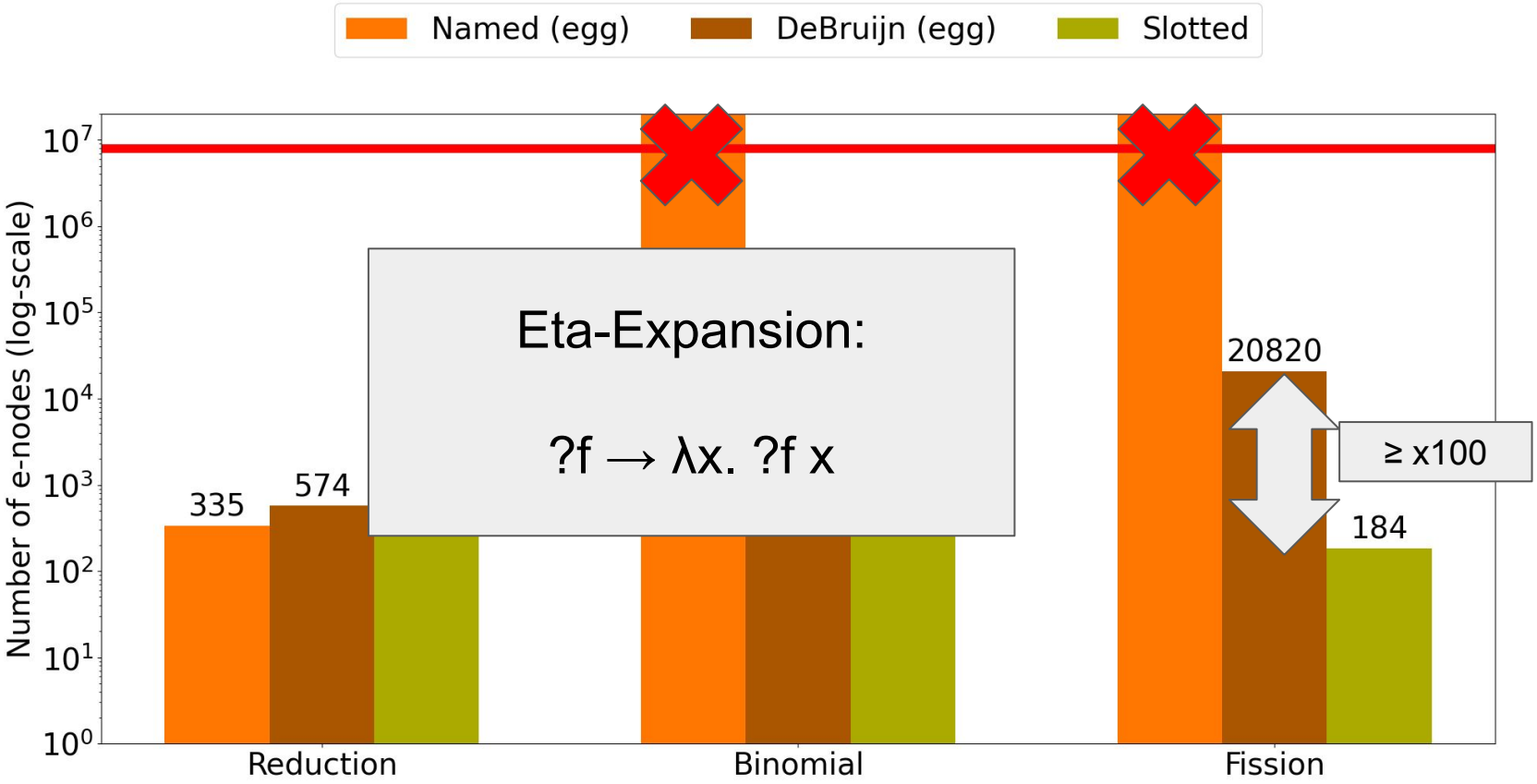
map ?f (map ?g ?arg) → map (λx. ?f (?g x)) ?arg

Map-fission:

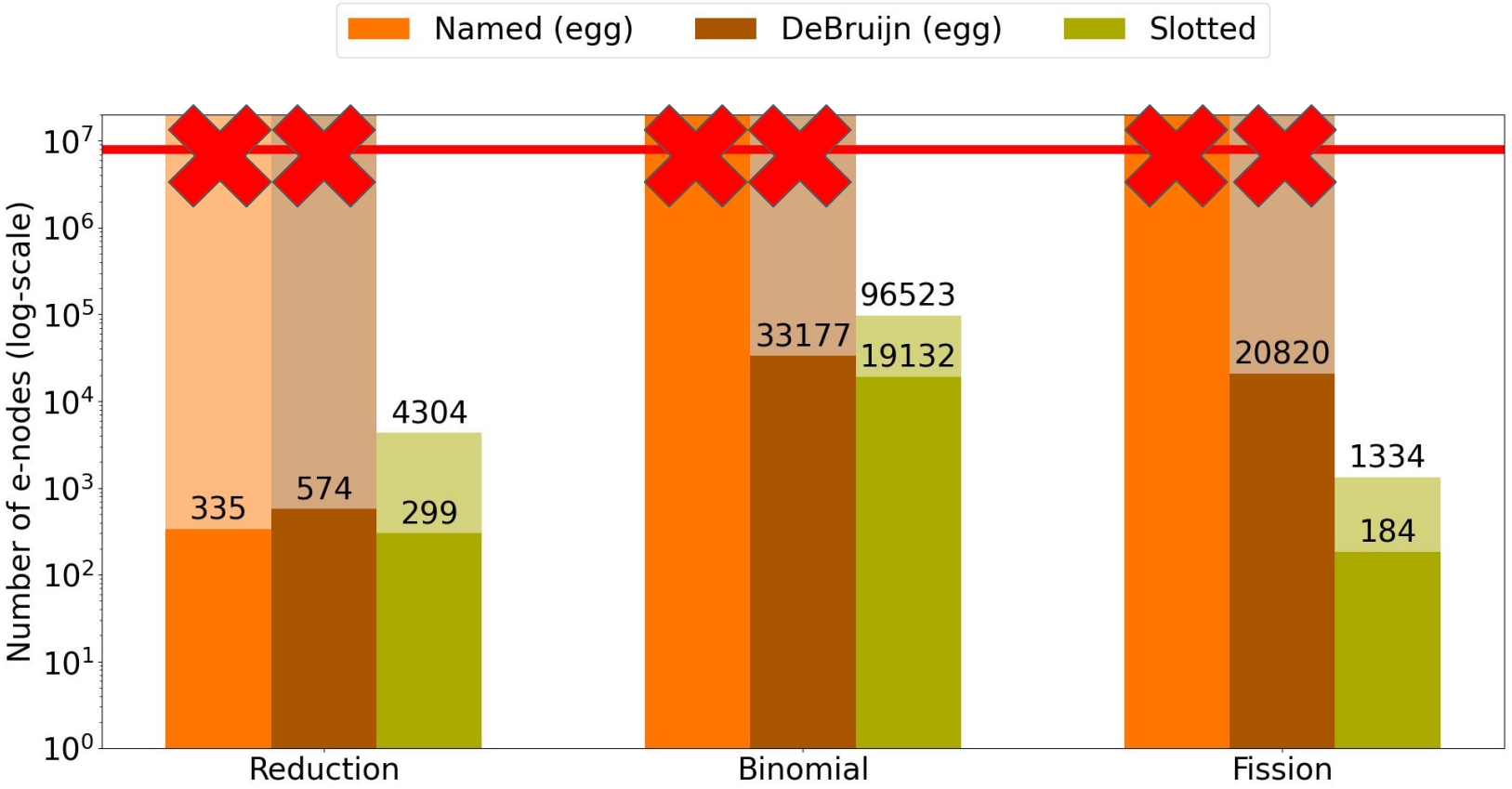
map (λx. ?f ?gx) → λy. map ?f (map (λx. ?gx) y),
if x not free in ?f.



Evaluation


Named (egg)
DeBruijn (egg)
Slotted
Number of e-nodes (log-scale)
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
335
574
20820
184
≥ x100
Eta-Expansion:
?f → λx. ?f x
Reduction
Binomial
Fission

Evaluation - with Eta-Expansion



State of the Implementation

- ✅ Specify your own Language
- ✅ E-Matching / Rewrite Rules
- ✅ Extraction (tree-only!)
- ✅ Challenge I: Parameter Mismatch
- ✅ Challenge II: Symmetry

- 🚧 No EGraph-Analysis yet
- 🚧 No fancy optimizations
- 🚧 Work-in-progress

https://github.com/memoryleak47/egraph-sandbox/tree/main/3-miniegg-with-slots