c.c.lemma: inductive proofs with e-graphs

Nadia Polikarpova

with Cole Kurashige, Ruyi Ji, Aditya Giridharan, Mark Barbone, Daniel Noor, Shachar Itzhaky, and Ranjit Jhala

WG2.8 2024

equations over recursive functions

```
half (add x x) ≐ x
```

equations over recursive functions

```
half (add x x) ≐ x
```

```
data Nat = Z | S Nat
```

```
add Z n         = n
add (S m') n    = S (add m' n)
```

```
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

example proof

```
half (add x x) ≐ x
```

```
data Nat = Z | S Nat
```

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

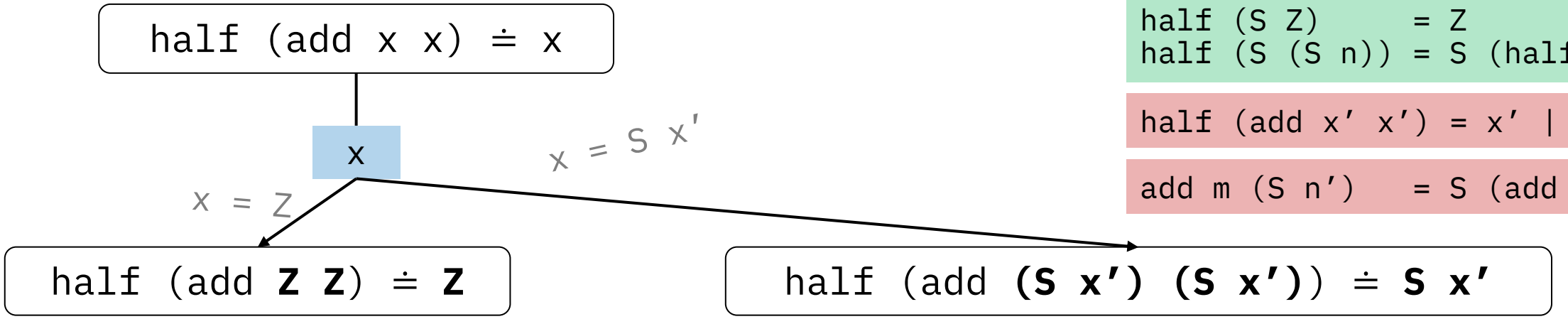
IH:

```
half (add x' x') = x' | x' ≺ x
```

lemma:

```
add m (S n')    = S (add m n')
```

example proof



⇨

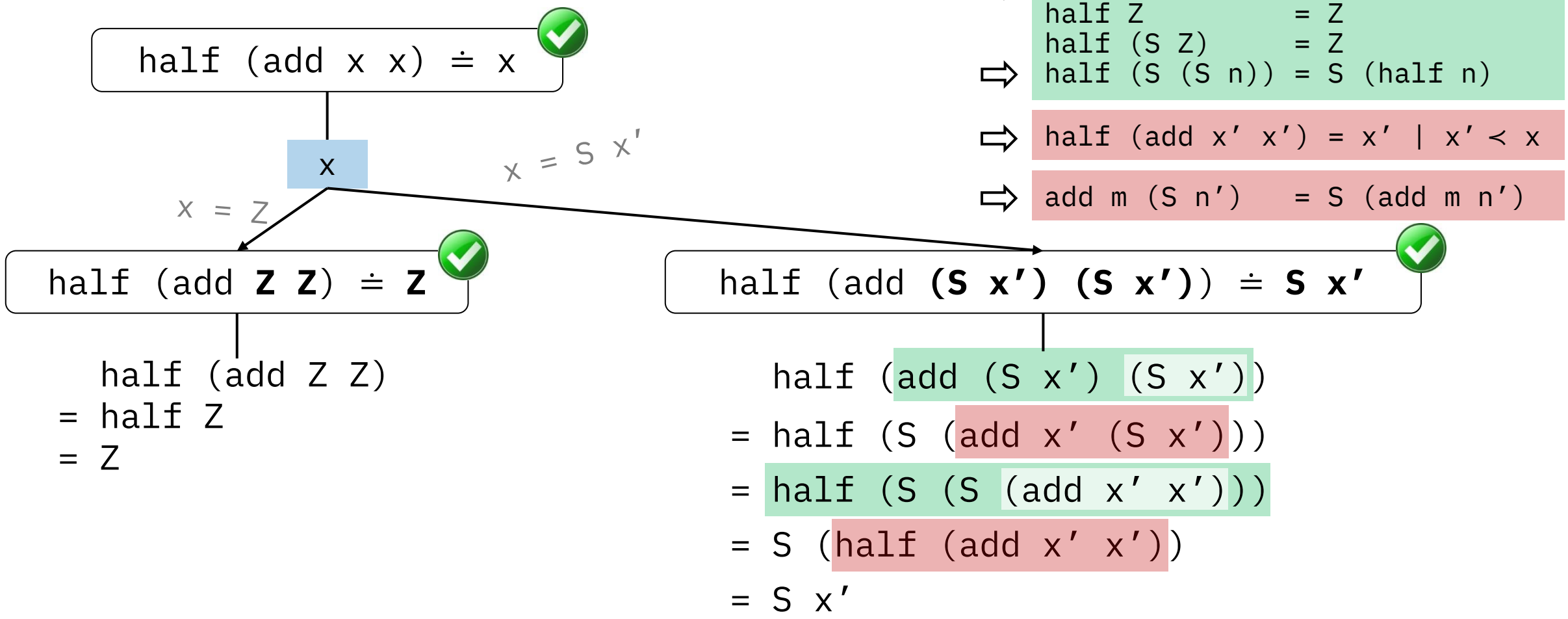
```
data Nat = Z | S Nat
```

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

```
half (add x' x') = x' | x' ≺ x
```

```
add m (S n')    = S (add m n')
```

example proof



this talk

1. two challenges
2. our solution: e-graphs
3. results

this talk

1. two challenges
    1. proof search
    2. lemma discovery
2. our solution: e-graphs
3. results

1. proof search

which equation to apply and where?

⇨
```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```
⇨ `half (add x' x') = x' | x' ≺ x`

⇨ `add m (S n')   = S (add m n')`

```
half (add (S x') (S x')) ≐ S x'
  half (add (S x') (S x'))
= half (S (add x' (S x')))
= half (S (S (add x' x')))
= S (half (add x' x'))
= S x'
```

1. proof search

which equation to apply and where?

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

⇨ `half (add x' x') = x' | x' ≺ x`

`add m (S n')   = S (add m n')`

`half (add (S x') (S x')) ≐ S x'`

```
  half (add (S x') (S x'))
= half (S (add x' (S x')))
= half (S (add (half (add x' x')) (S x')))
= half (S (add (half (add (half (add x' x')) x')) (S x')))
= …
```

1. proof search

which equation to apply and where?

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

```
half (add x' x') = x' | x' ≺ x
```

```
add m (S n')    = S (add m n')
```

```
half (add (S x') (S x')) ≐ S x'
```

past solution: backtracking search

2. lemma discovery

where did we get this lemma from?

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

```
half (add x' x') = x' | x' ≺ x
```

⇨

```
add m (S n')    = S (add m n')
```

past solutions?

```
half (add (S x') (S x')) ≐ S x'
```

```
  half (add (S x') (S x'))
= half (S (add x' (S x')))
= half (S (S (add x' x')))
= S (half (add x' x'))
= S x'
```

past solutions to lemma discovery

generalization

generalize the goal

+ proving lemma ⇒ success
+ scales well
− incomplete

examples:

- IsaPlanner [Dixon & Fleuriot 2004]
- Zeno [Sonnex at al 2012]

theory exploration

enumerate and check all lemmas

− lemma might not apply anywhere in the proof!
− scales poorly with size of lemma and vocab
+ complete

examples:

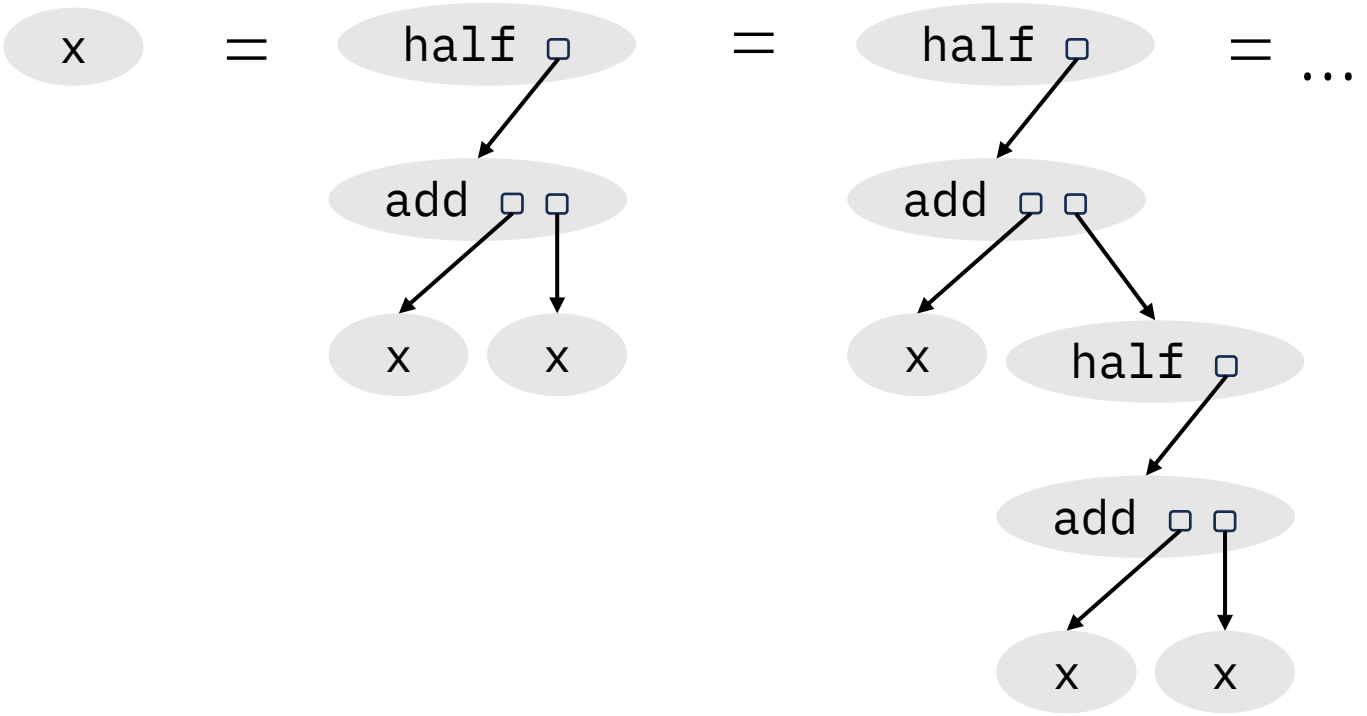
- HipSpec [Claessen et al 2013]
- CVC4 [Reynolds & Kuncak 2015]
- TheSy [Singher & Itzhaky 2021]

this talk

1. two challenges
2. our solution: e-graphs
3. results

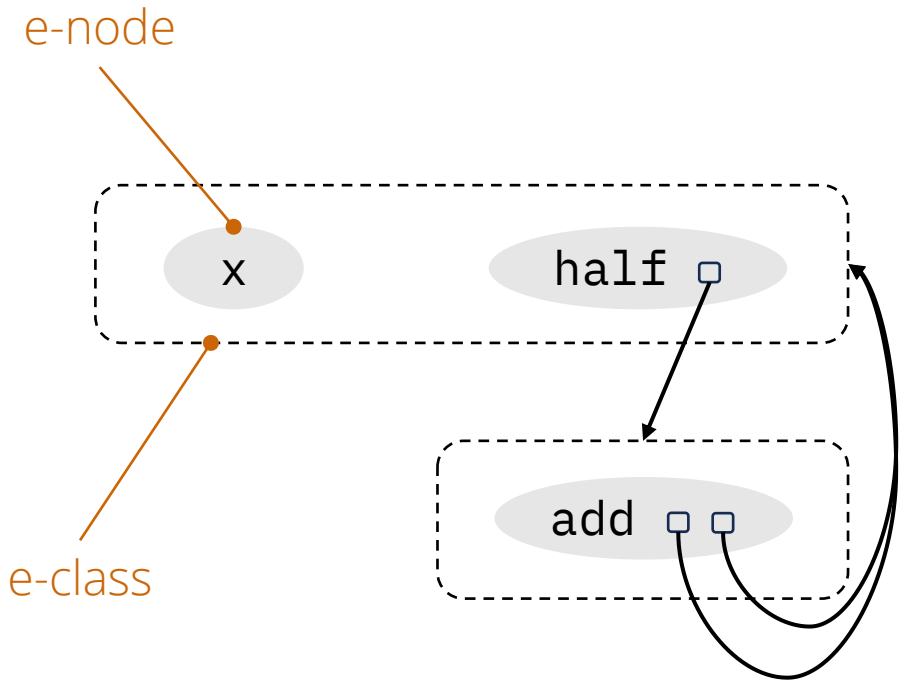
e-graphs

compactly represent
sets of equivalent terms



e-graphs

compactly represent
sets of equivalent terms

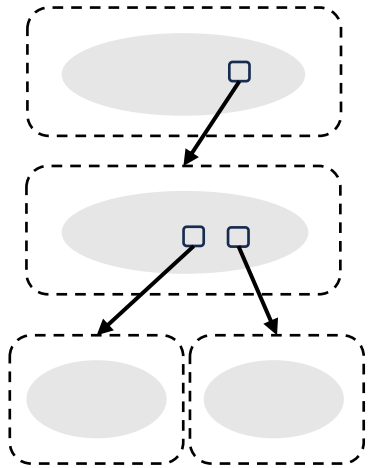


equality saturation

[Tate et al. POPL'09]

egg [Willsey et al. POPL'21]

e-graph / term

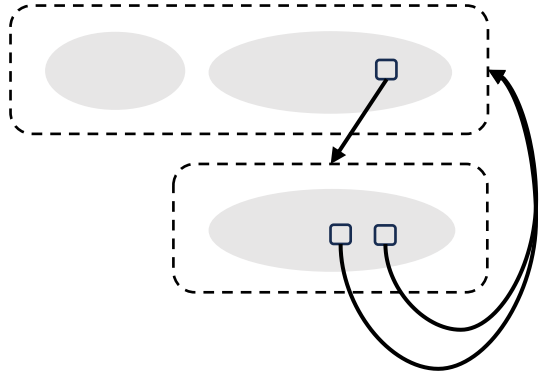


saturate

with equations / rewrites

```
half (add x x) = x
…
```

saturated e-graph



e-classes =
equivalence classes wrt rewrites

1. proof search with e-graphs

```
half (add (S x') (S x')) ≐ S x'
```

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

```
half (add x' x') = x' | x' < x
add m (S n')    = S (add m n')
```

which equation to apply and where?

our solution:
every equation, everywhere, all at once!

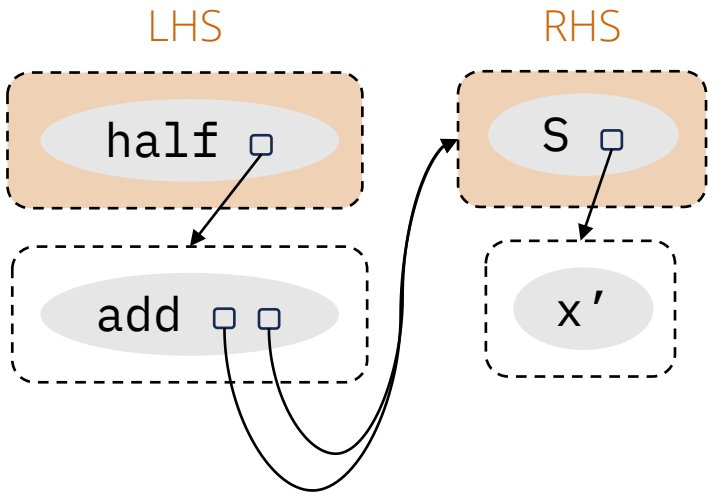
1. proof search with e-graphs

```
half (add (S x') (S x')) ≐ S x'
```

```
add Z n         = n
add (S m') n    = S (add m' n)
half Z          = Z
half (S Z)      = Z
half (S (S n)) = S (half n)
```

```
half (add x' x') = x' | x' < x
add m (S n')    = S (add m n')
```

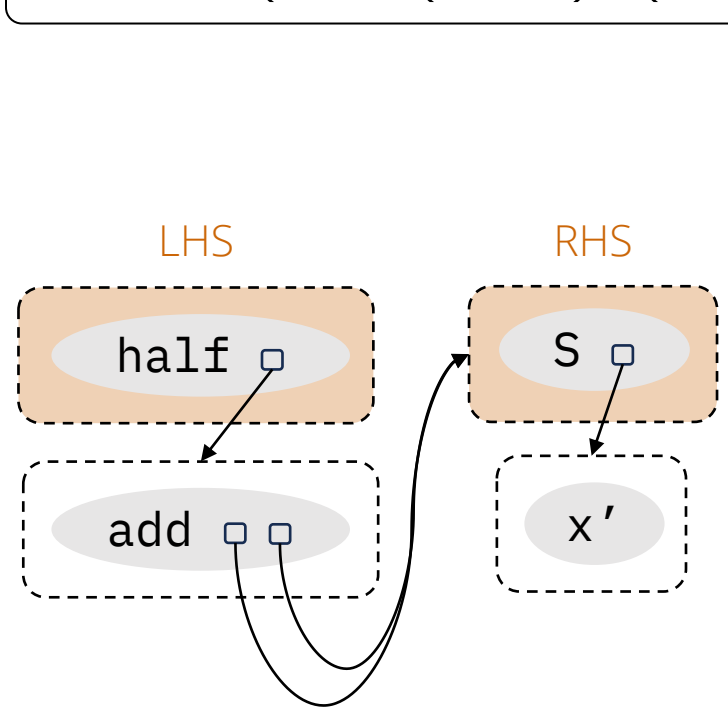
step 1: throw LHS and RHS into an e-graph



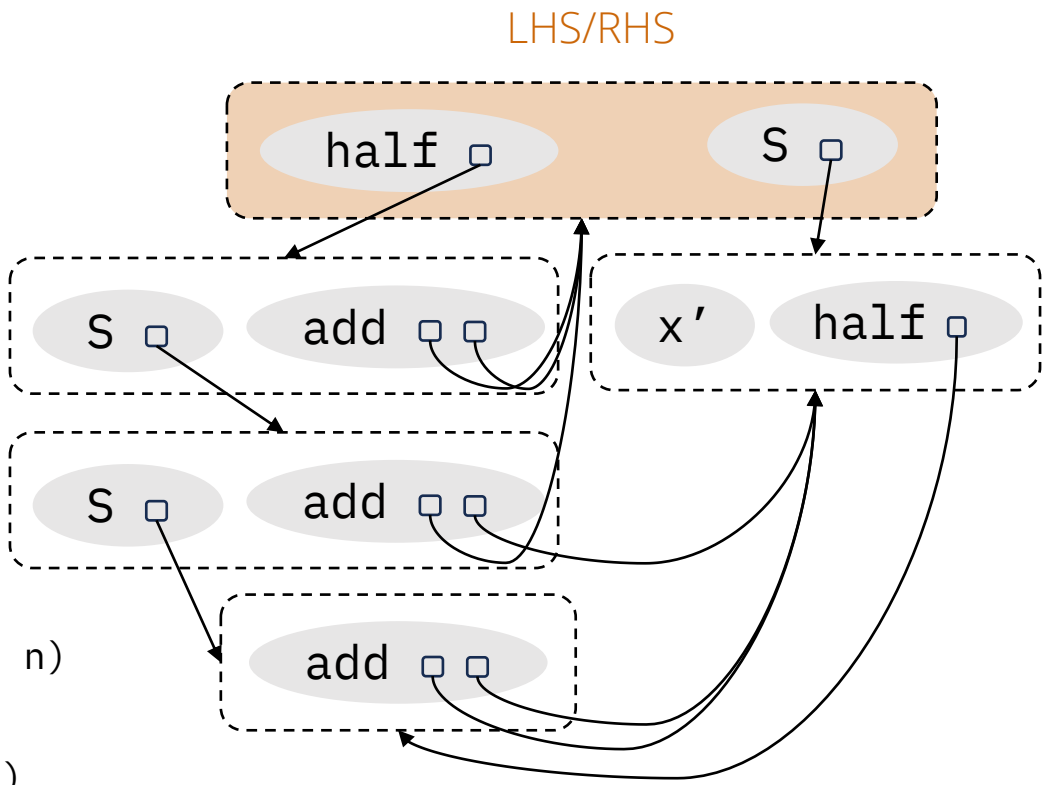
step 2: saturate

1. proof search with e-graphs

```
half (add (S x') (S x')) ≐ S x'
```



step 2: saturate

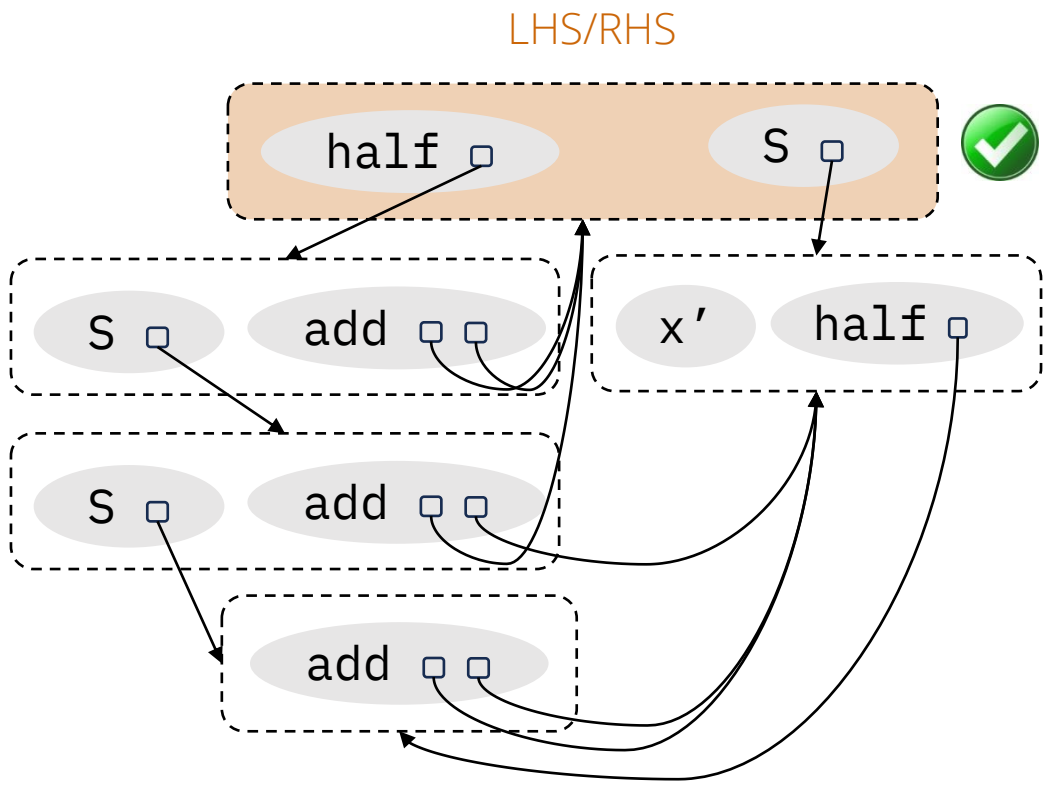



```
add Z n          = n
add (S m') n     = S (add m' n)
half Z           = Z
half (S Z)       = Z
half (S (S n))   = S (half n)
half (add x' x') = x' | x' ≺ x
add m (S n')     = S (add m n')
```

1. proof search with e-graphs

```
half (add (S x') (S x')) ≐ S x'
```

step 3: check if LHS and RHS are in the same e-class



2. lemma discovery

where did we get this lemma from?

```
add Z n          = n
add (S m') n     = S (add m' n)
half Z           = Z
half (S Z)       = Z
half (S (S n))   = S (half n)
```

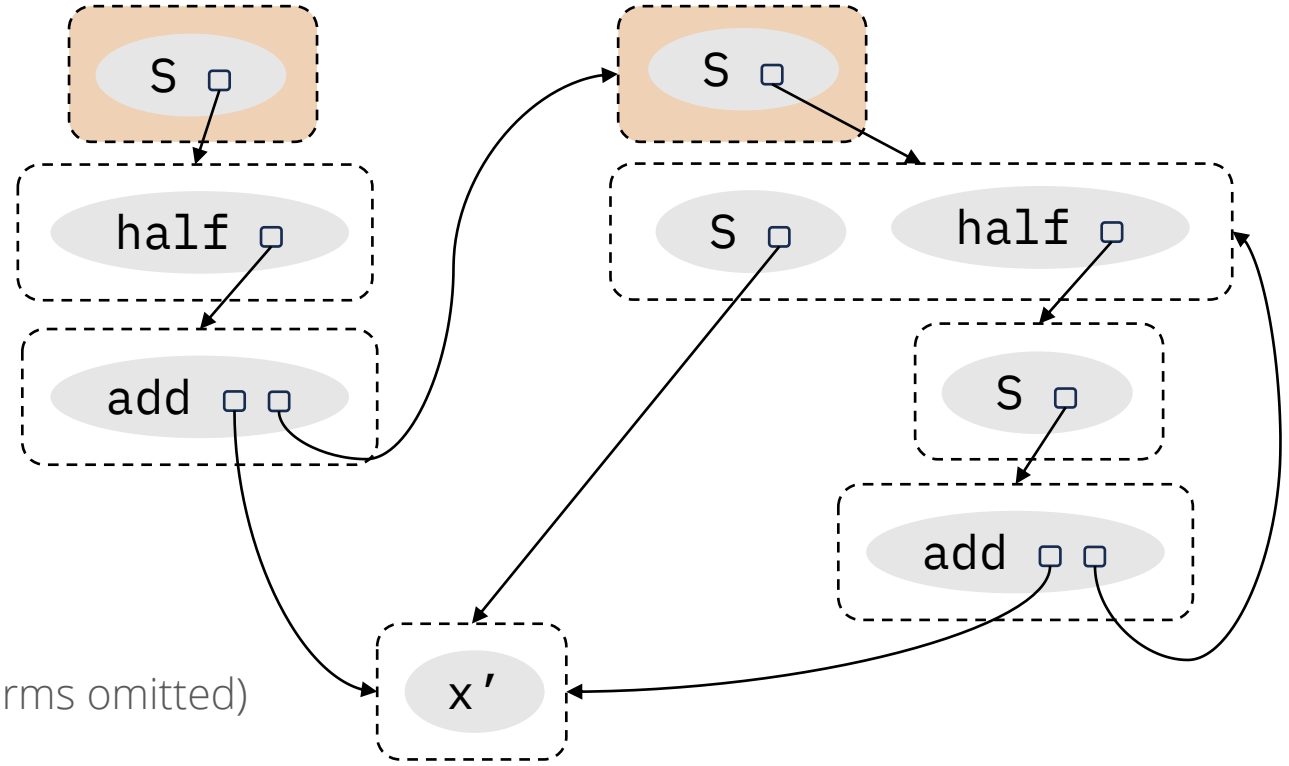
⇨

```
half (add x' x') = x' | x' ≺ x
add m (S n')     = S (add m n')
```

our solution:
extract from the e-graph!

2. lemma discovery with e-graphs

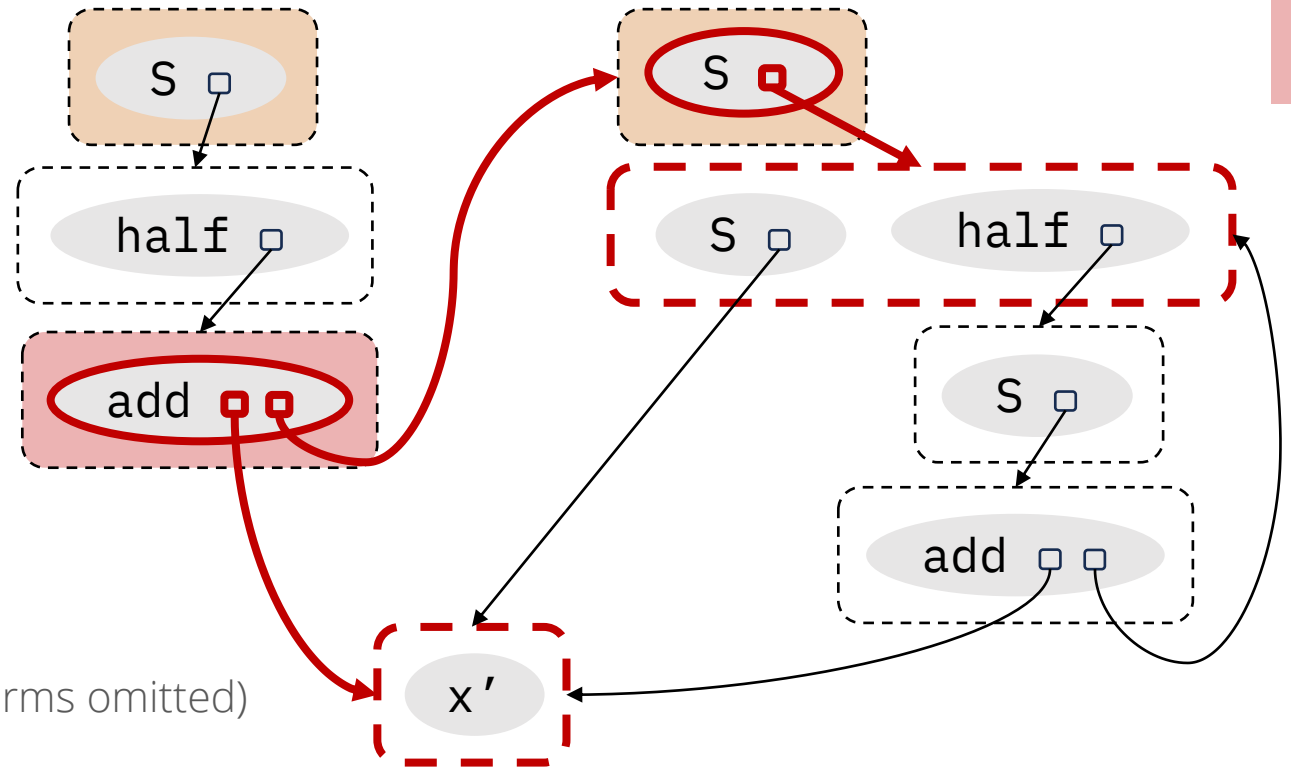
`half (add (S (S x'')) (S (S x''))) ≐ S (S x'')`



(reducible terms omitted)

2. lemma discovery with e-graphs

`half (add (S (S x'')) (S (S x''))) ≐ S (S x'')`

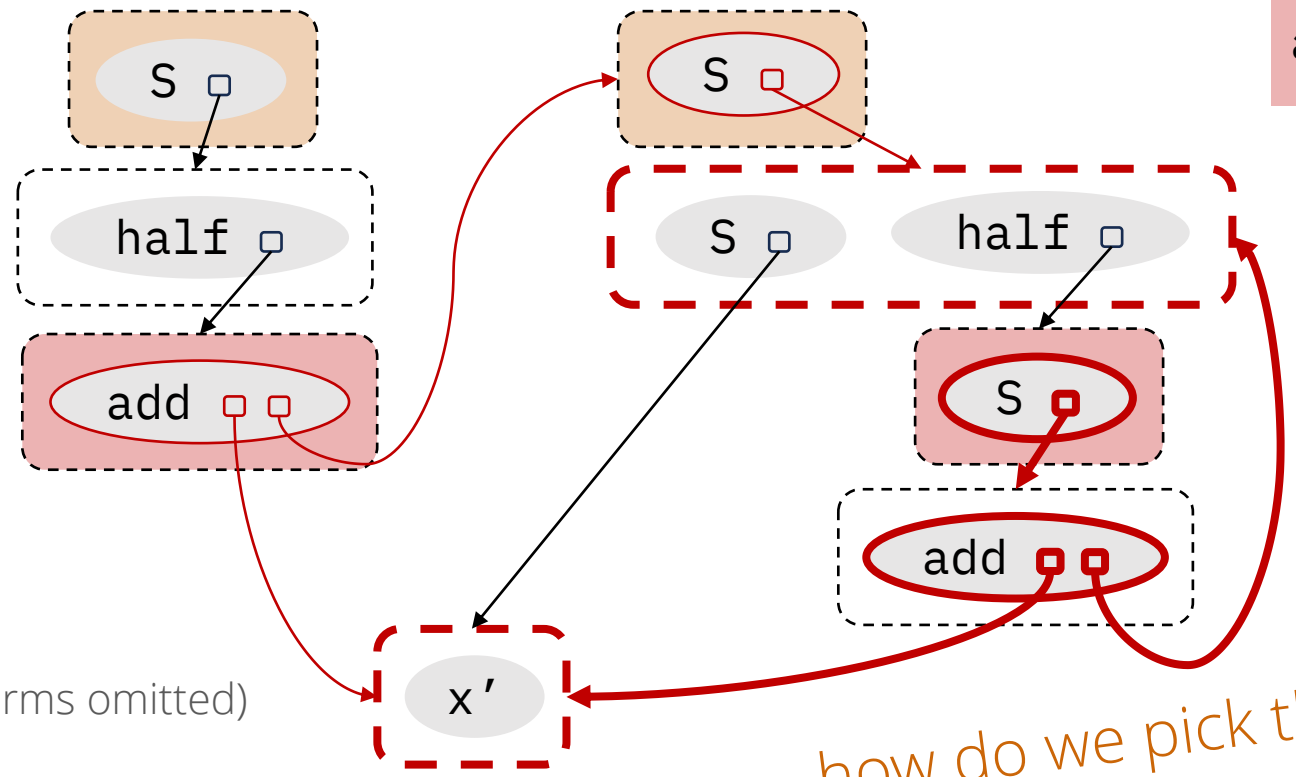


`add m (S n')`

(reducible terms omitted)

2. lemma discovery with e-graphs

`half (add (S (S x'')) (S (S x''))) ≐ S (S x'')`

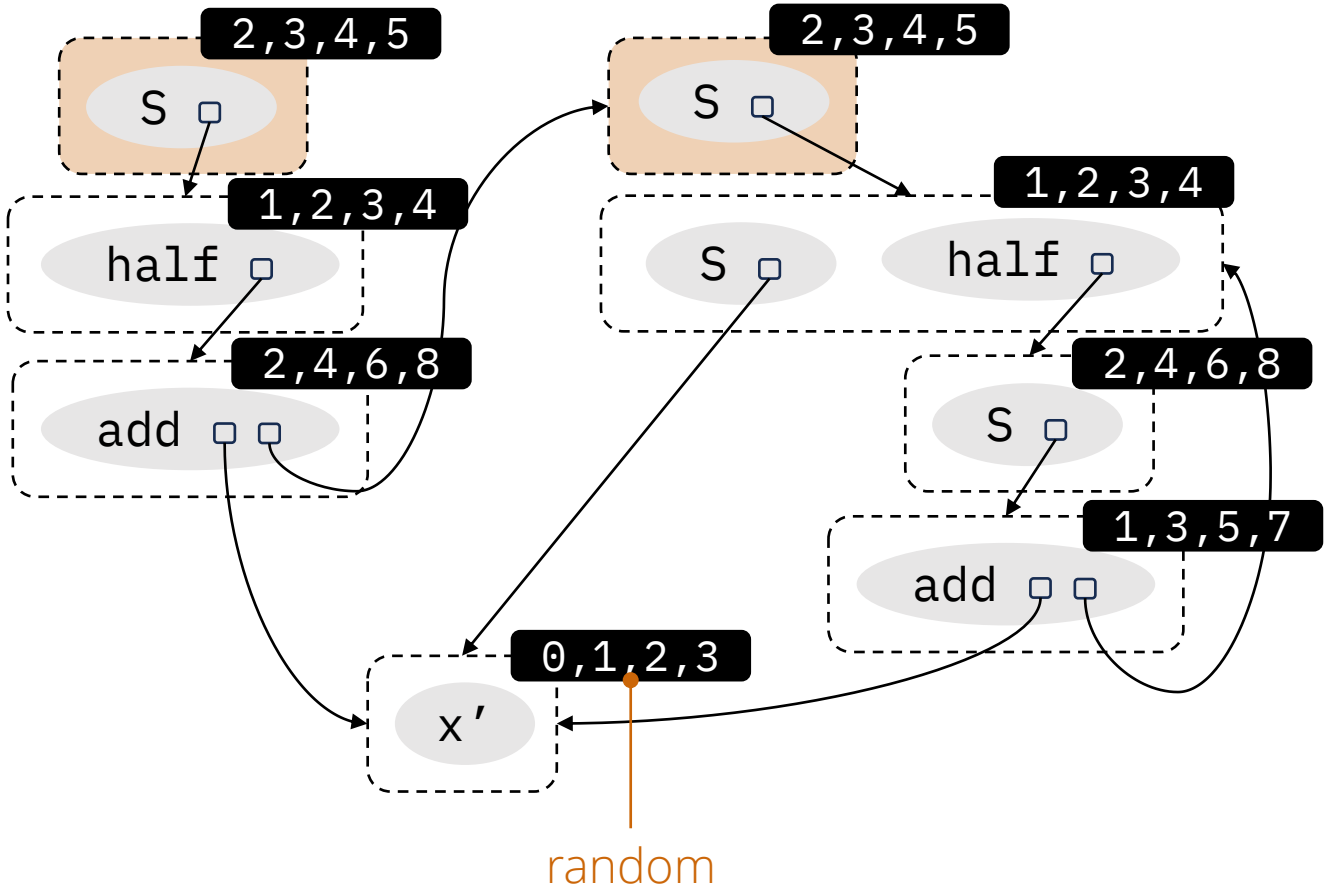


`add m (S n') ≟ S (add m n')`

(reducible terms omitted)

how do we pick the two candidate e-classes?

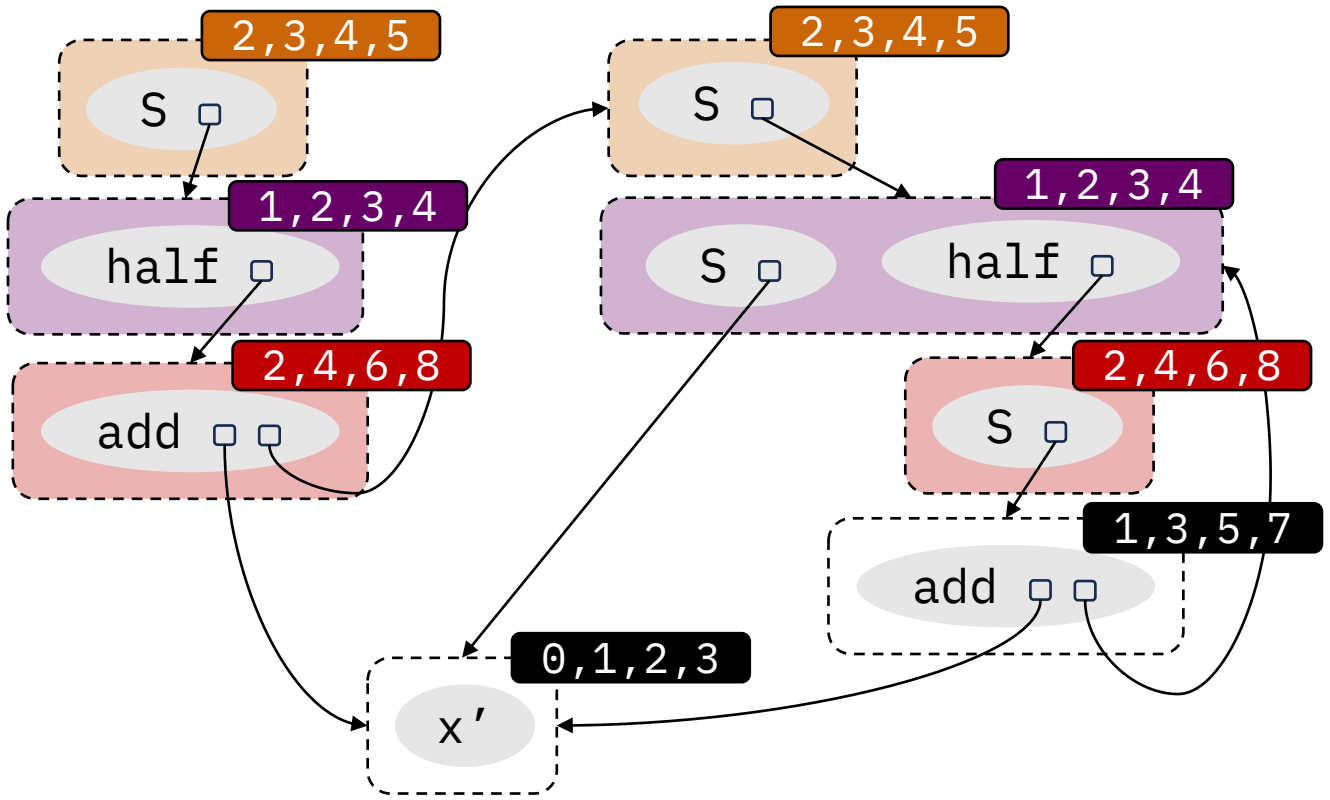
filtering with c-vecs



Prior work:
filter out invalid lemmas via testing

Ruler [Nandi et al 2021]:
can be done efficiently in an e-graph by incrementally maintaining characteristic vectors (c-vecs)

filtering with c-vecs



Prior work:
filter out invalid lemmas via testing

Ruler [Nandi et al 2021]:
can be done efficiently in an e-graph
by incrementally maintaining
characteristic vectors (c-vecs)

only 3 pairs of e-classes to extract lemmas from!

c.c.lemma

uses e-graphs to efficiently

1. perform proof search without backtracking
2. discover lemmas that unify existing e-classes

compared to prior lemma discovery methods:

generalization	c.c.lemma
+ proving lemma ⇒ success	− proving lemma ⇏ success
+ scales well	+ scales better
− incomplete	+ more complete

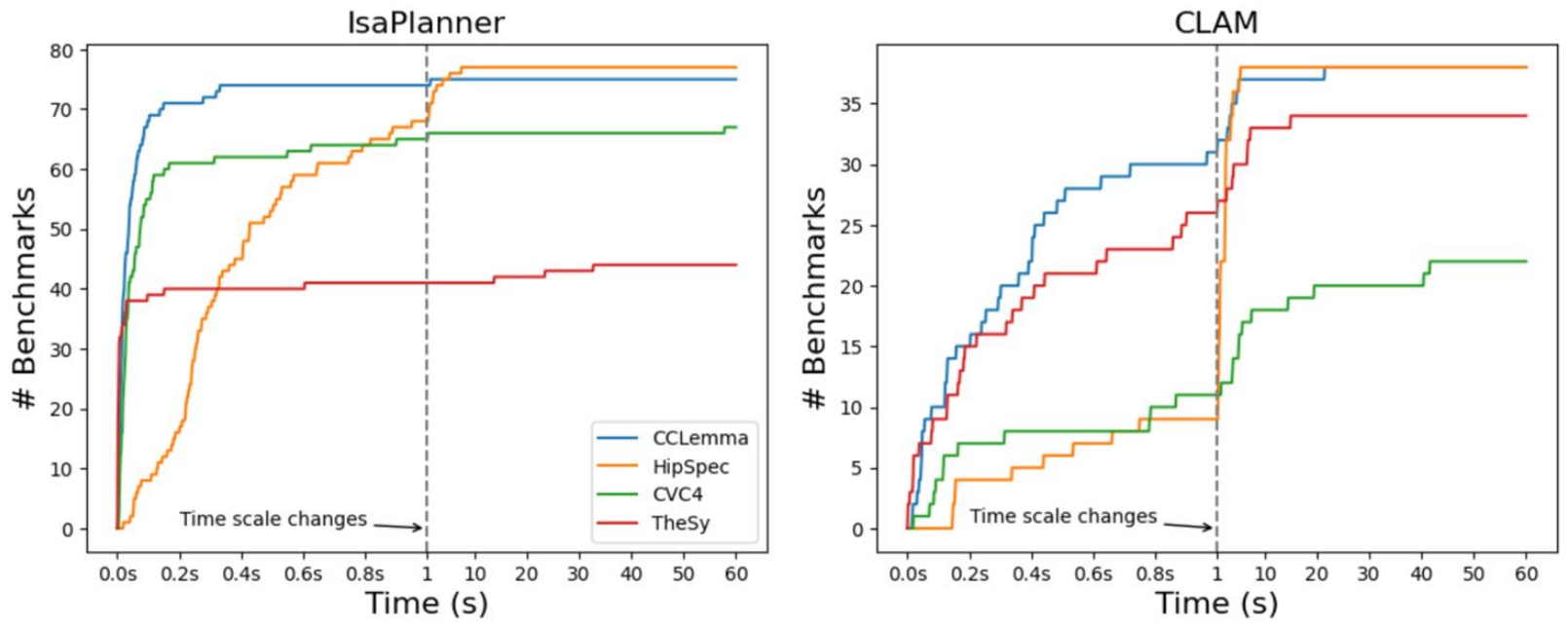
c.c.lemma	theory exploration
+ proving lemma makes progress	− lemma might not apply anywhere
+ scales well	− scales poorly
− incomplete	+ complete

this talk

1. two challenges
2. our solution: e-graphs
3. results

“easy” benchmarks: IsaPlaner and CLAM

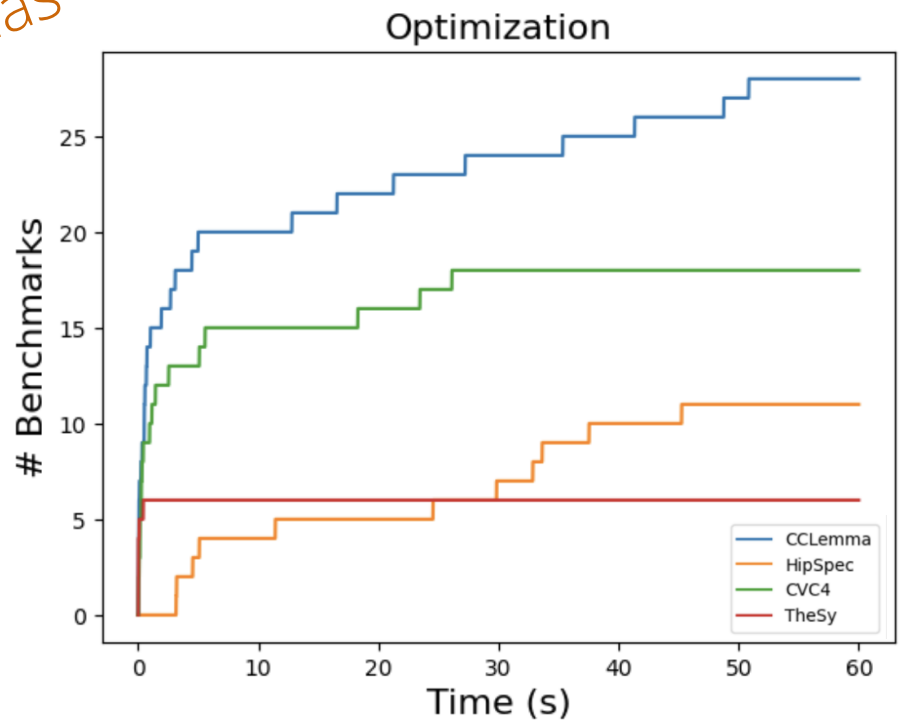
```
half (add x x) ≐ x

drop n (drop m xs) ≐ drop (add n m) xs
```



hard benchmark: optimization

```
maximum (map product (tails xs)) ≐
fst (foldr (\x (res,prod) -> (max res (prod*x), prod*x)) (1,1) xs)
```

large vocab & large lemmas



generalization	c.c.lemma
+ proving lemma ⇒ success + scales well − incomplete	− proving lemma ⇏ success + scales better + more complete

c.c.lemma	theory exploration
+ proving lemma makes progress + scales well − incomplete	− lemma might not apply anywhere − scales poorly + complete

https://github.com/cole-k/cc-lemma