(a * 2) / 2

…

= a

$(a * 2) / 2$

$= a * (2 / 2)$

$= a * 1$

$= a$

The secrets of *arbor syntaxis reducto*

```
(a * b) / c ⇄ a * (b / c)

a / a ⇄ 1

a * 1 ⇄ a

a * 2 ⇄ a << 1
```

The secrets of *arbor syntaxis reducto*

```
(a * b) / c ⇄ a * (b / c)

a / a ⇄ 1

a * 1 ⇄ a

a * 2 ⇄ a << 1
```

a * 1 ⇄ a

a

a * 1

a * 1 * 1

a * 1 * 1 * 1 …

The secrets of *arbor syntaxis reducto*

```
(a * b) / c ⇄ a * (b / c)

a / a ⇄ 1

a * 1 ⇄ a

a * 2 ⇄ a << 1
```

The secrets of *arbor syntaxis reducto*

```
a * 2 ⇄ a << 1
```

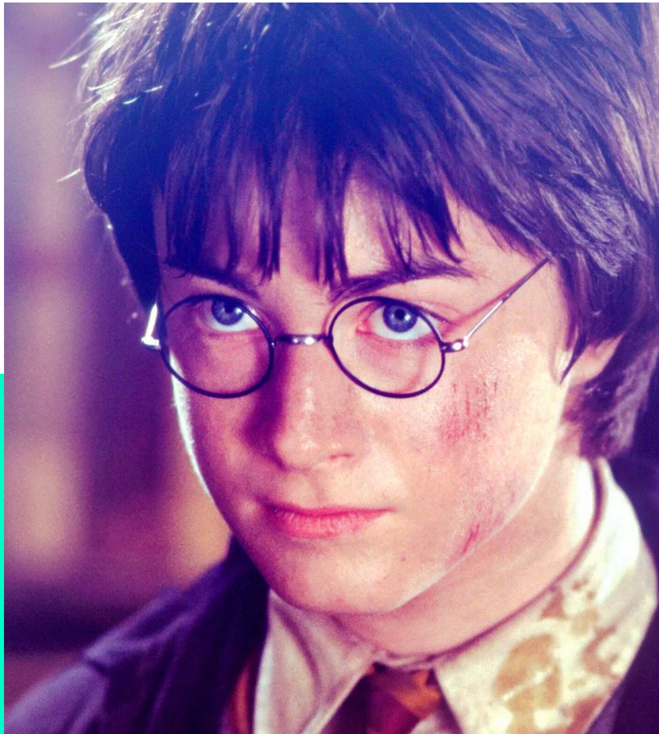
```
(a * 2) / 2
(a << 1) / 2 …
what now?
```

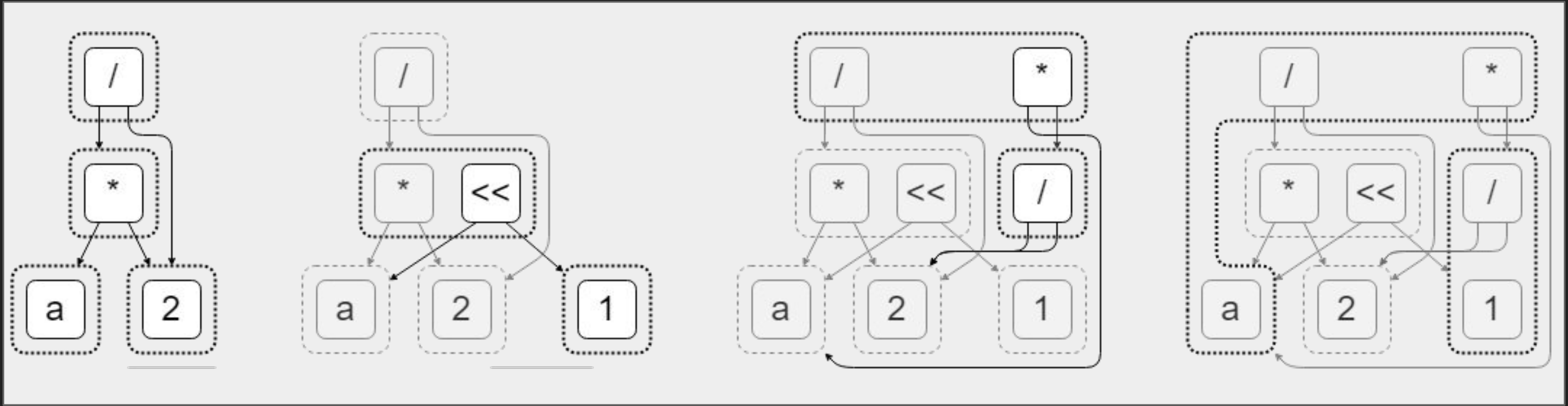
```
a * 2 ⇄ a << 1
```

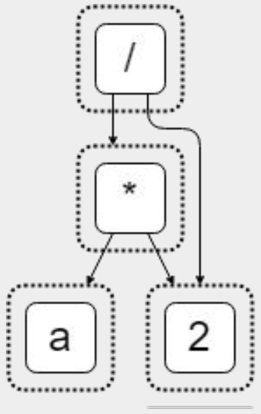
#lang-talk
PICTURES
A TimeWarner Company

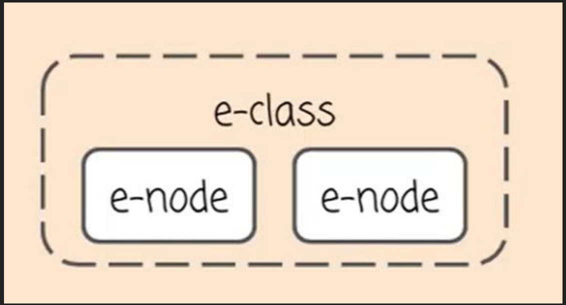
Harry Potter and The Methods of Equality Saturation

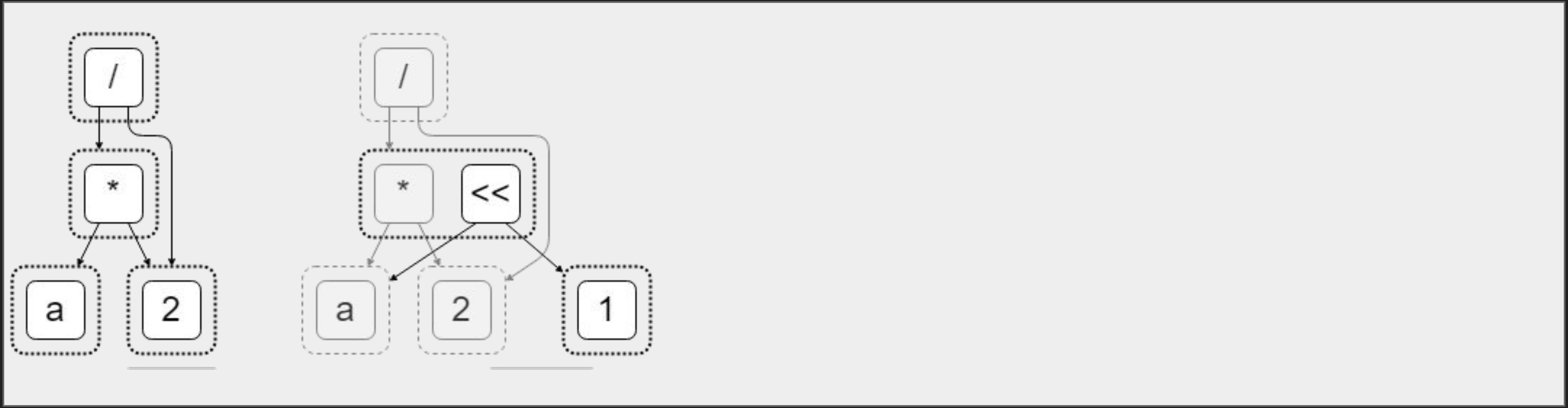
they see me rowling…



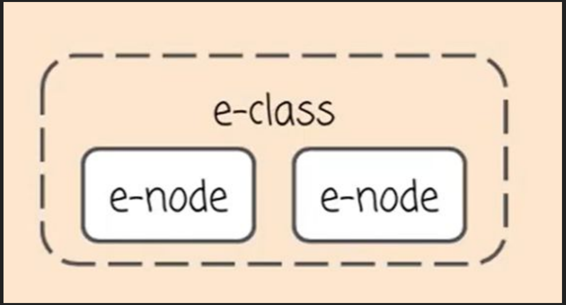

/
*
a
2
<<
1

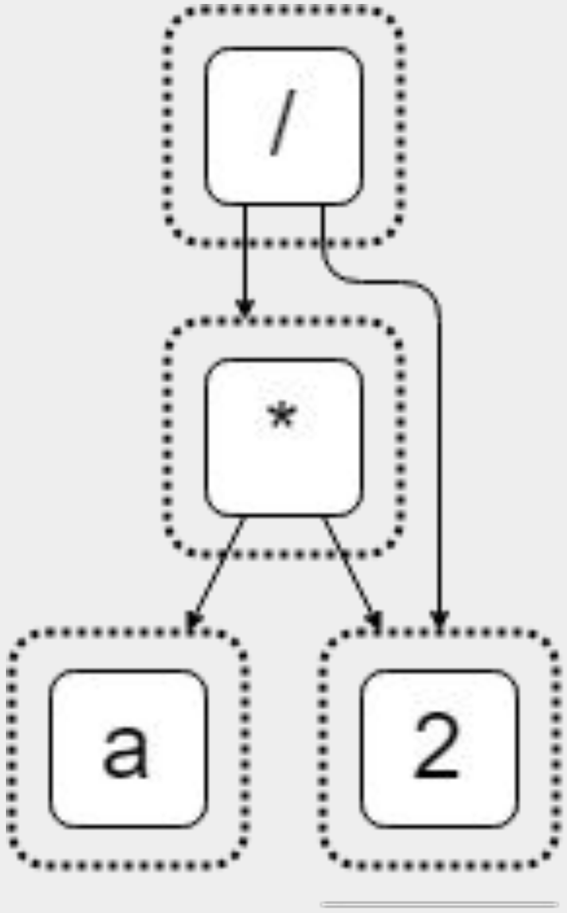

/
*
a
2

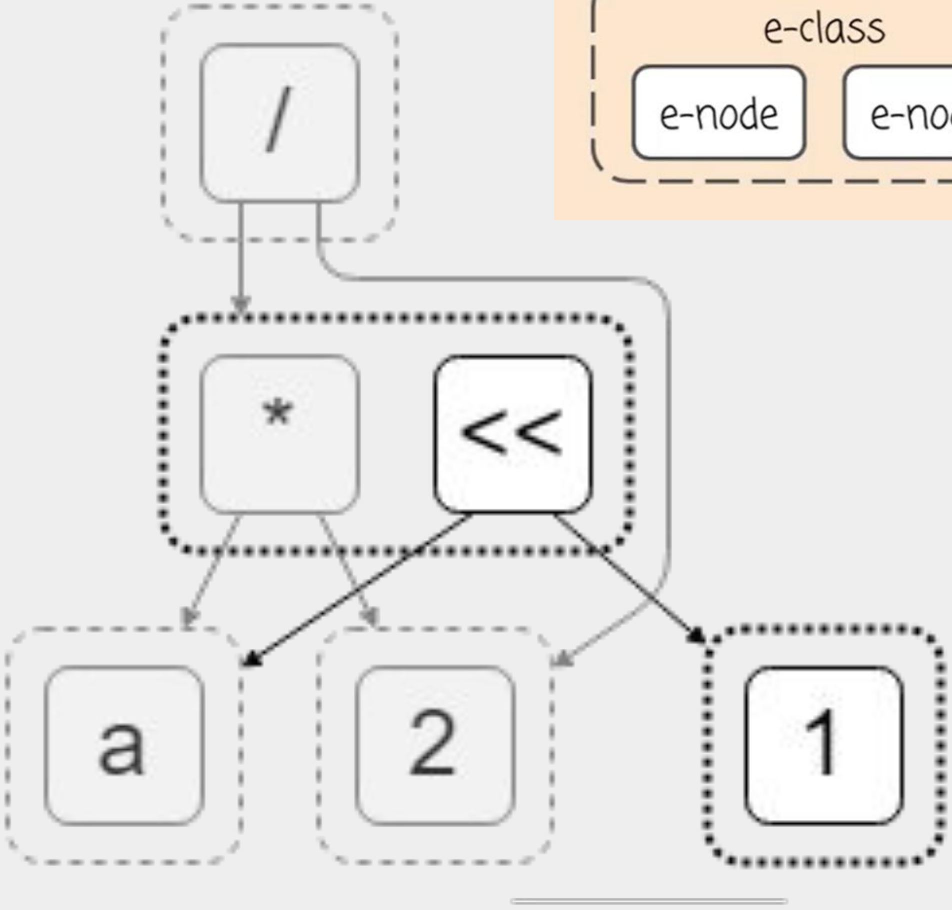

e-class
e-node
e-node

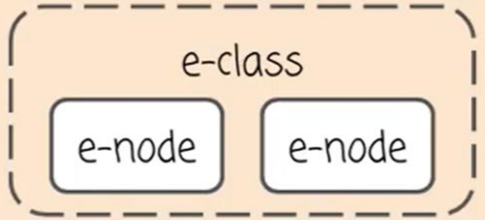


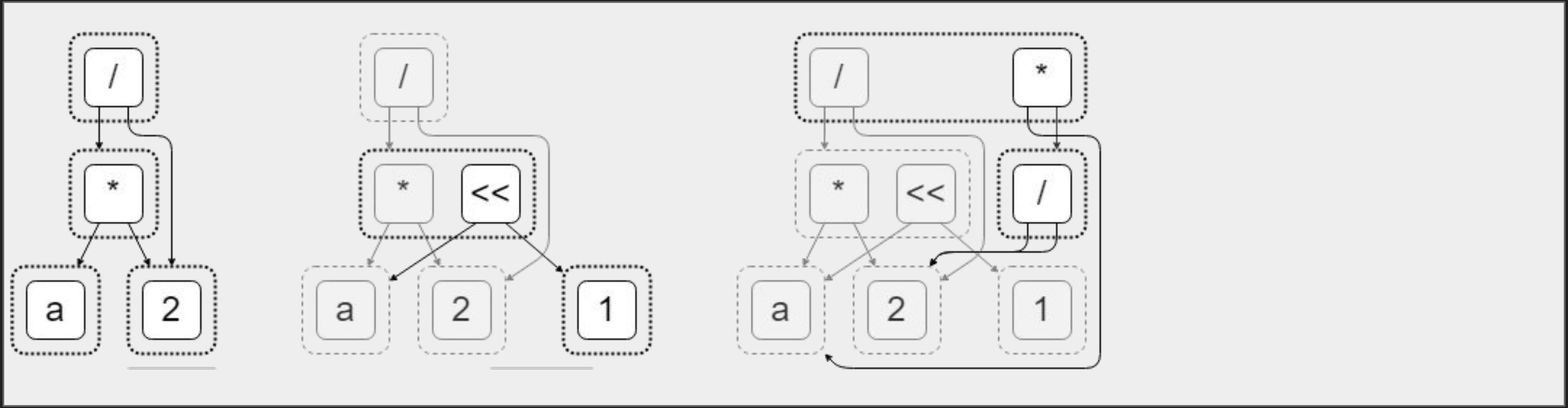
a * 2 → a << 1



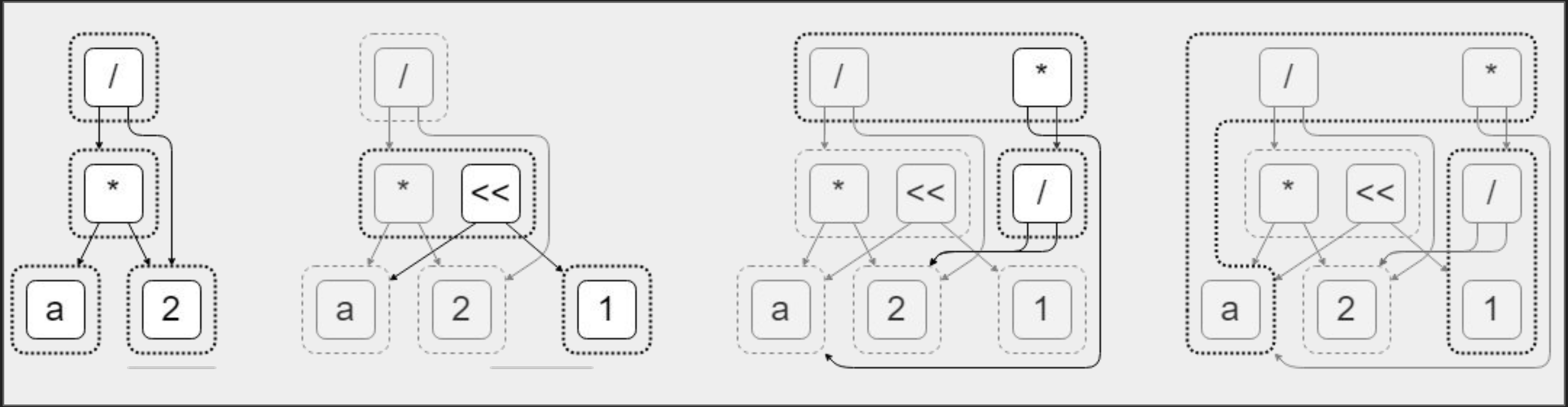

/
*
a
2


/
*
<<
a
2
1


e-class
e-node
e-node



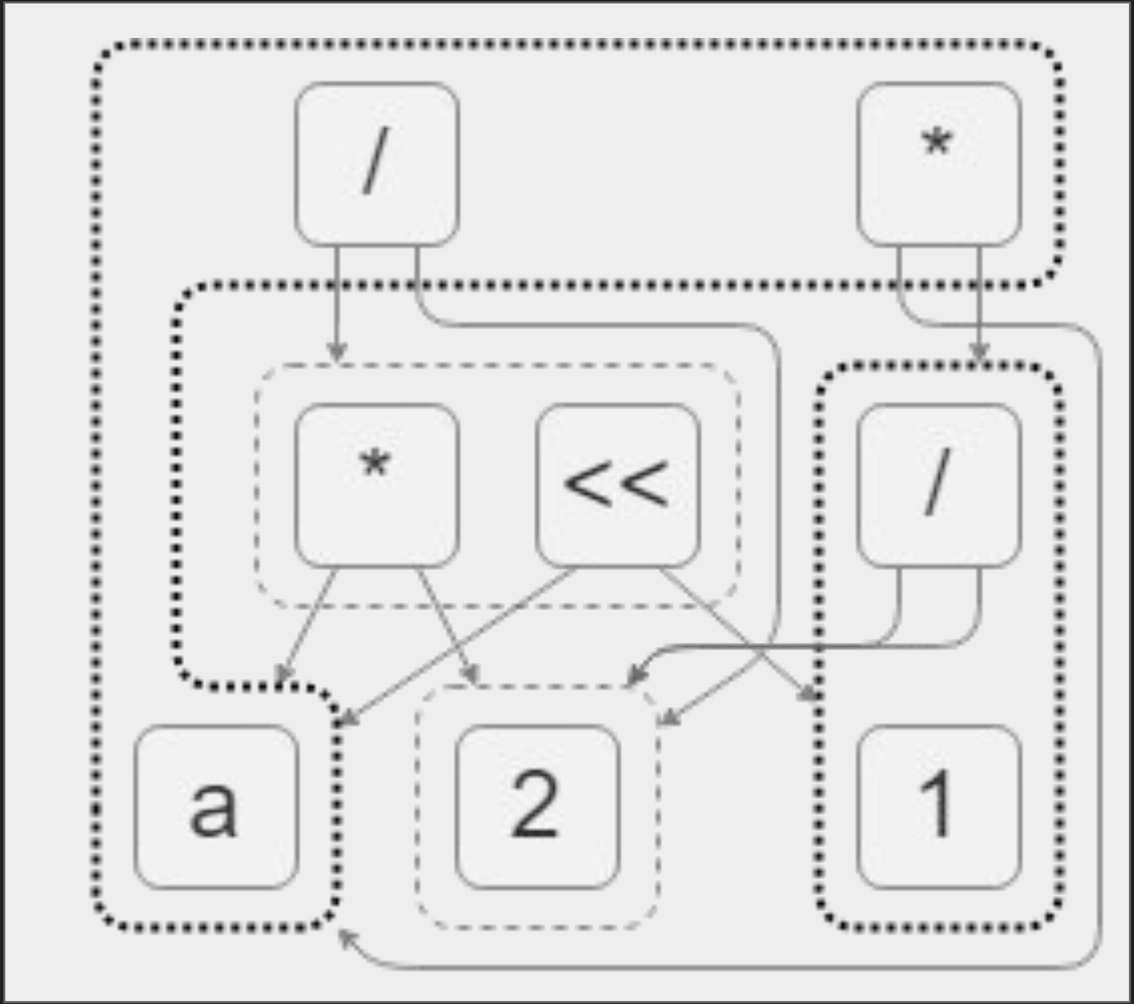
(a * 2) / 2 → a * (2 / 2)



2 / 2 → 1

a * 1 → a

```
a
a * 1
a * 1 * 1
a * 1 * 1 * 1 …
```

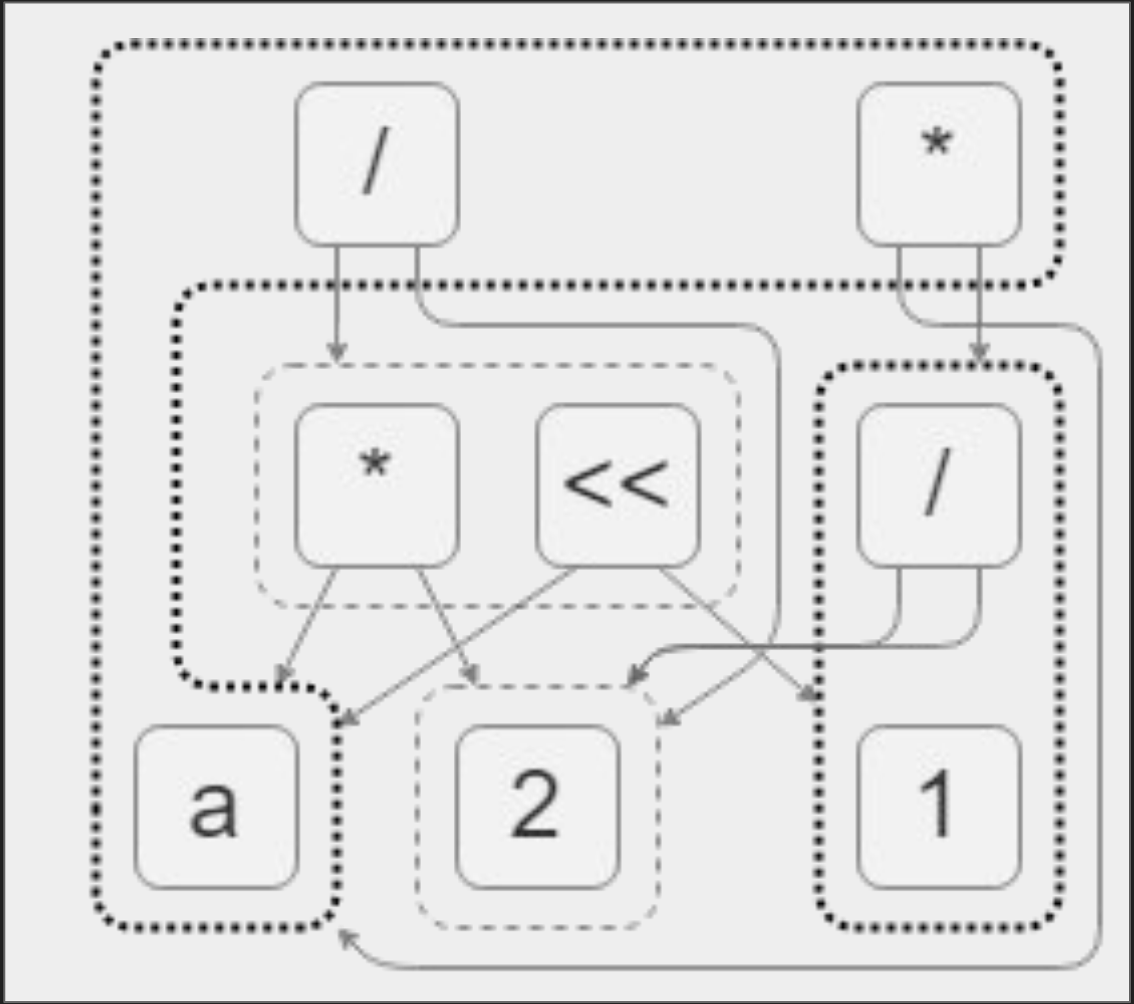


in just

4 e-classes!

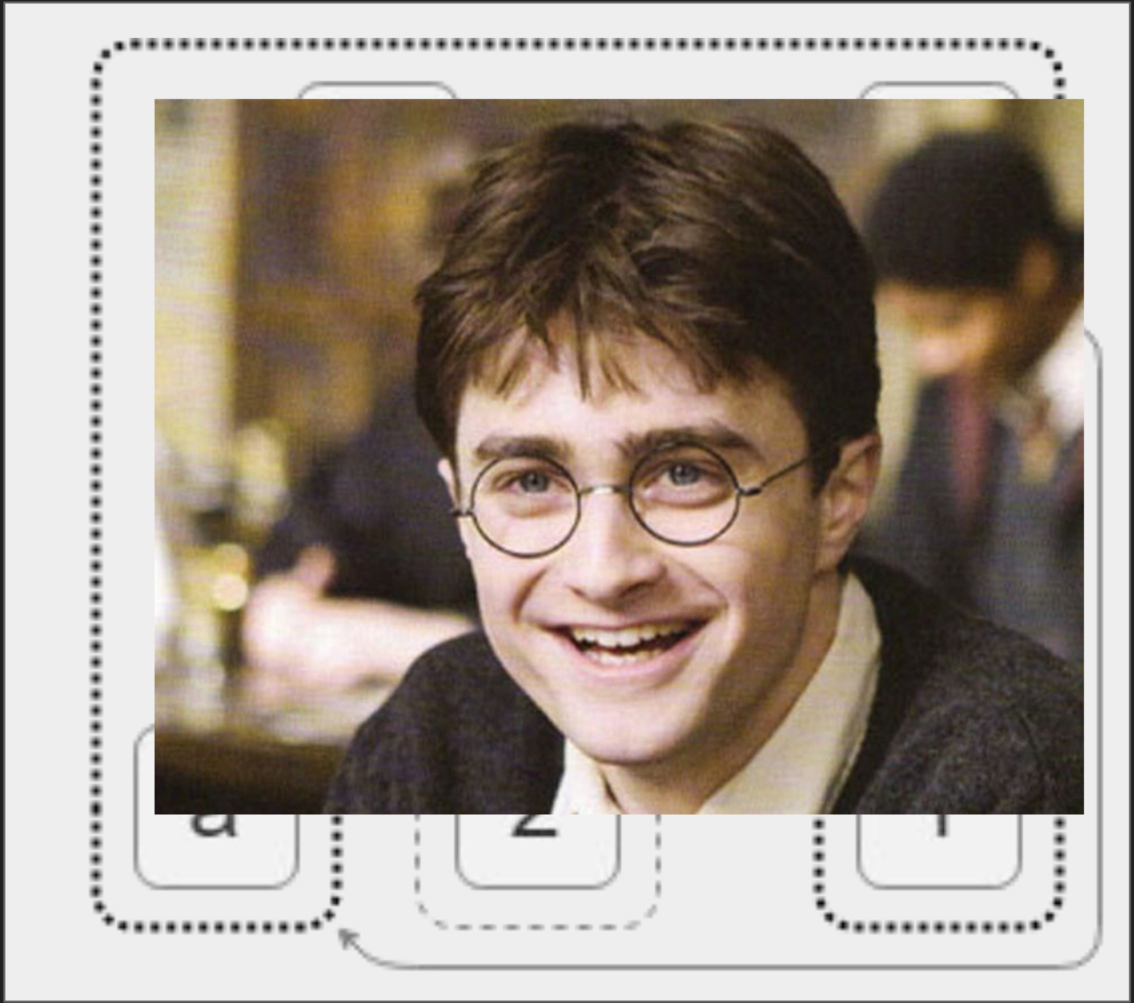
```
a
a * 1
a * 1 * 1
a * 1 * 1 * 1 …
```

and it’s **saturated!**



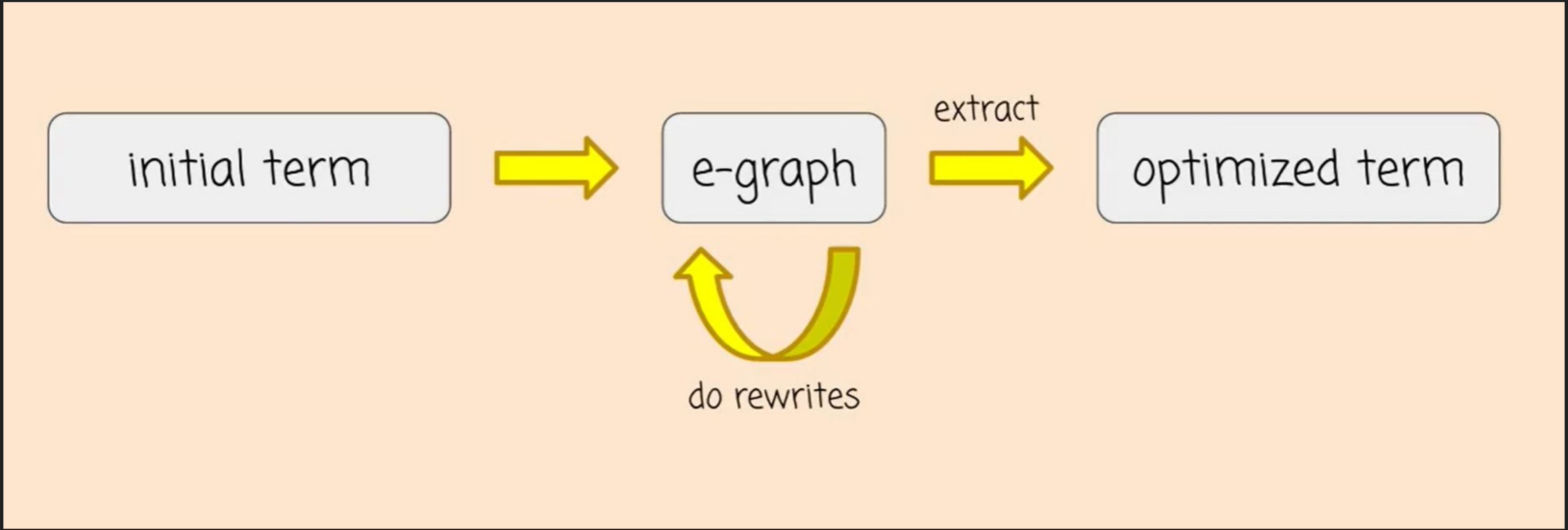
```
a
a * 1
a * 1 * 1
a * 1 * 1 * 1 …
```

and it’s **saturated!**

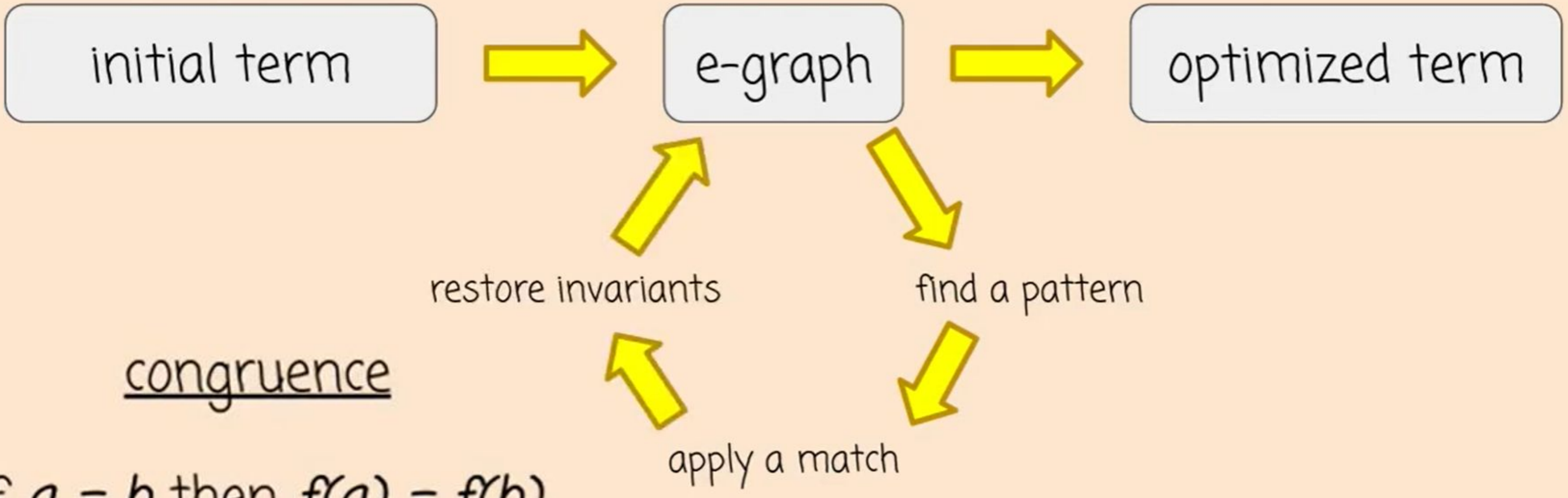


egg: e-graphs good

Not the end of the line just yet



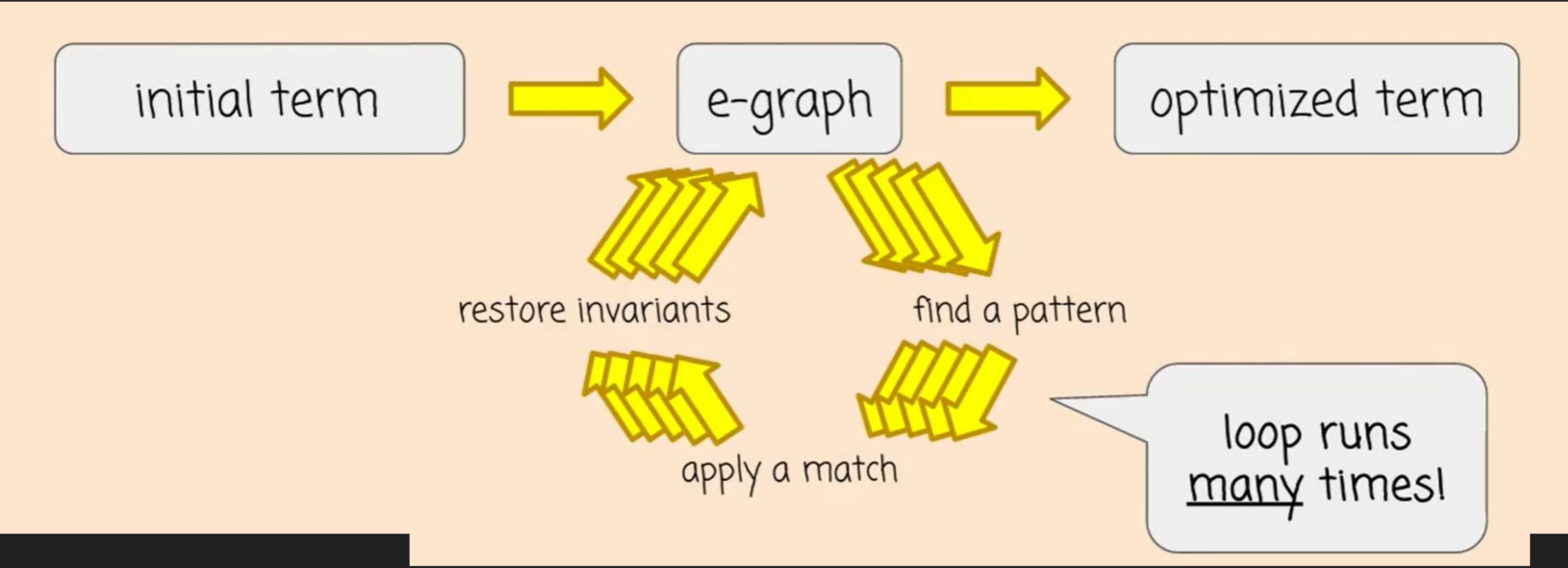
Not the end of the line just yet



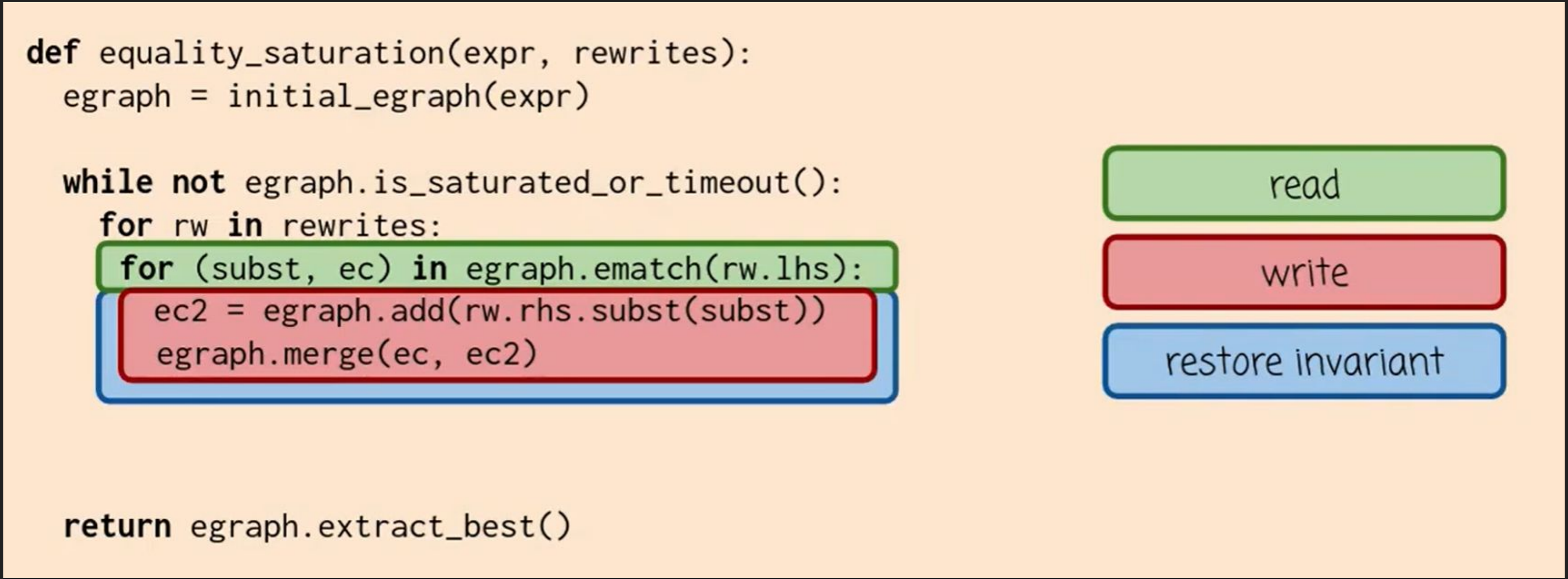
<u>congruence</u>

if $a = b$ then $f(a) = f(b)$

Not the end of the line just yet

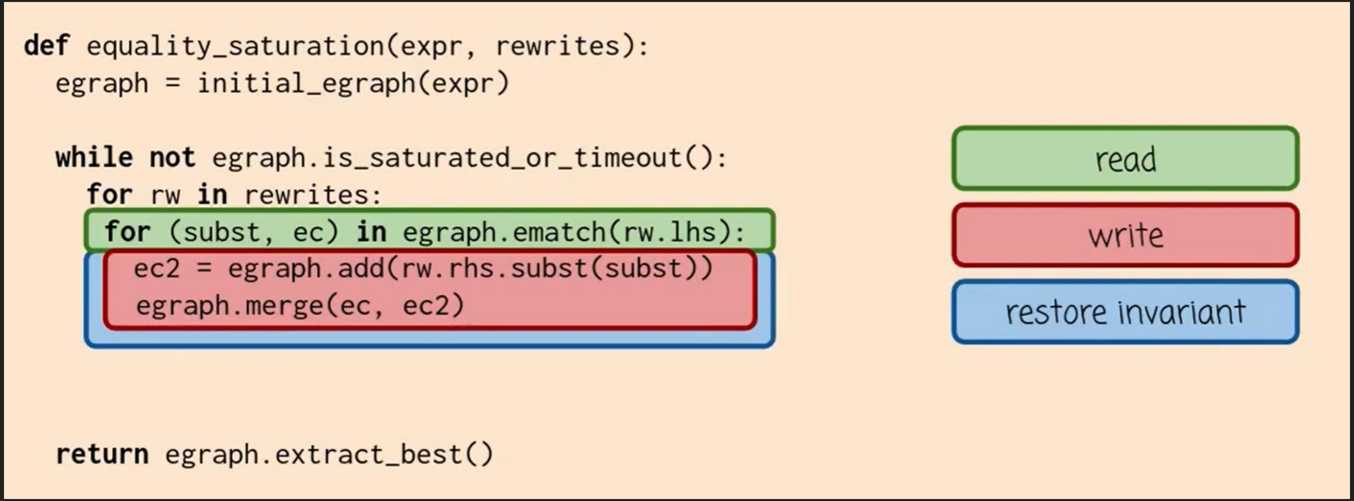


Equality saturation



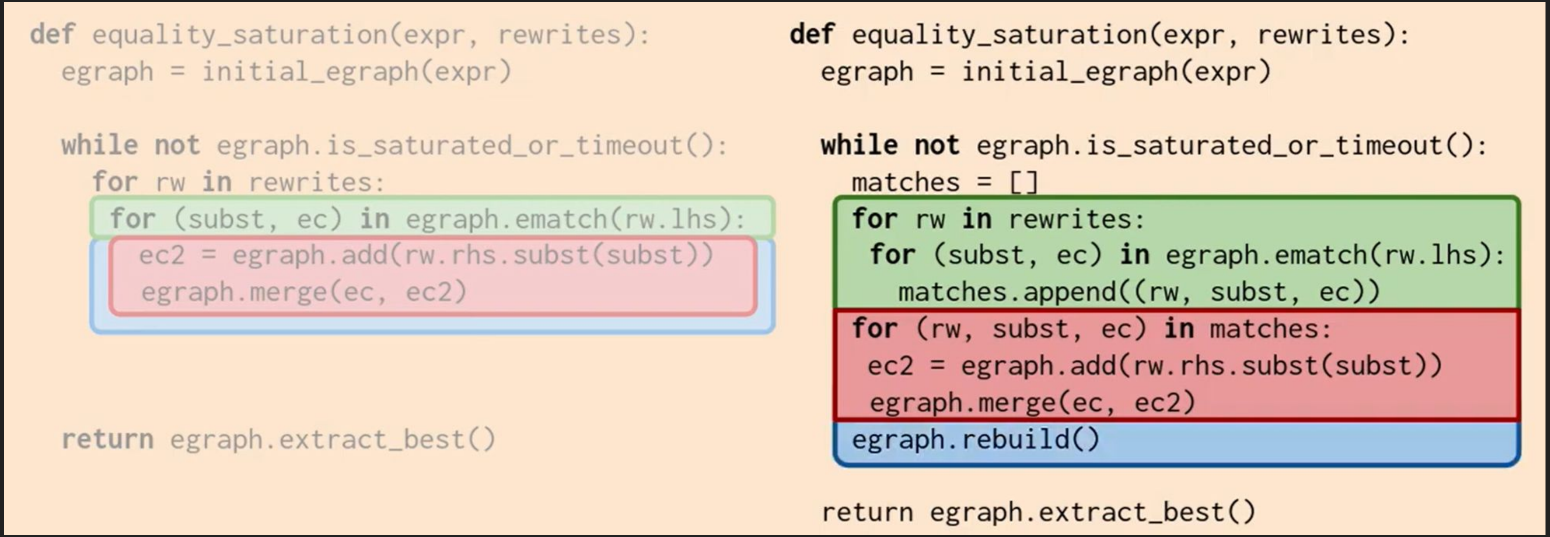
```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(expr)

  while not egraph.is_saturated_or_timeout():
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        ec2 = egraph.add(rw.rhs.subst(subst))
        egraph.merge(ec, ec2)



  return egraph.extract_best()
```

Equality saturation



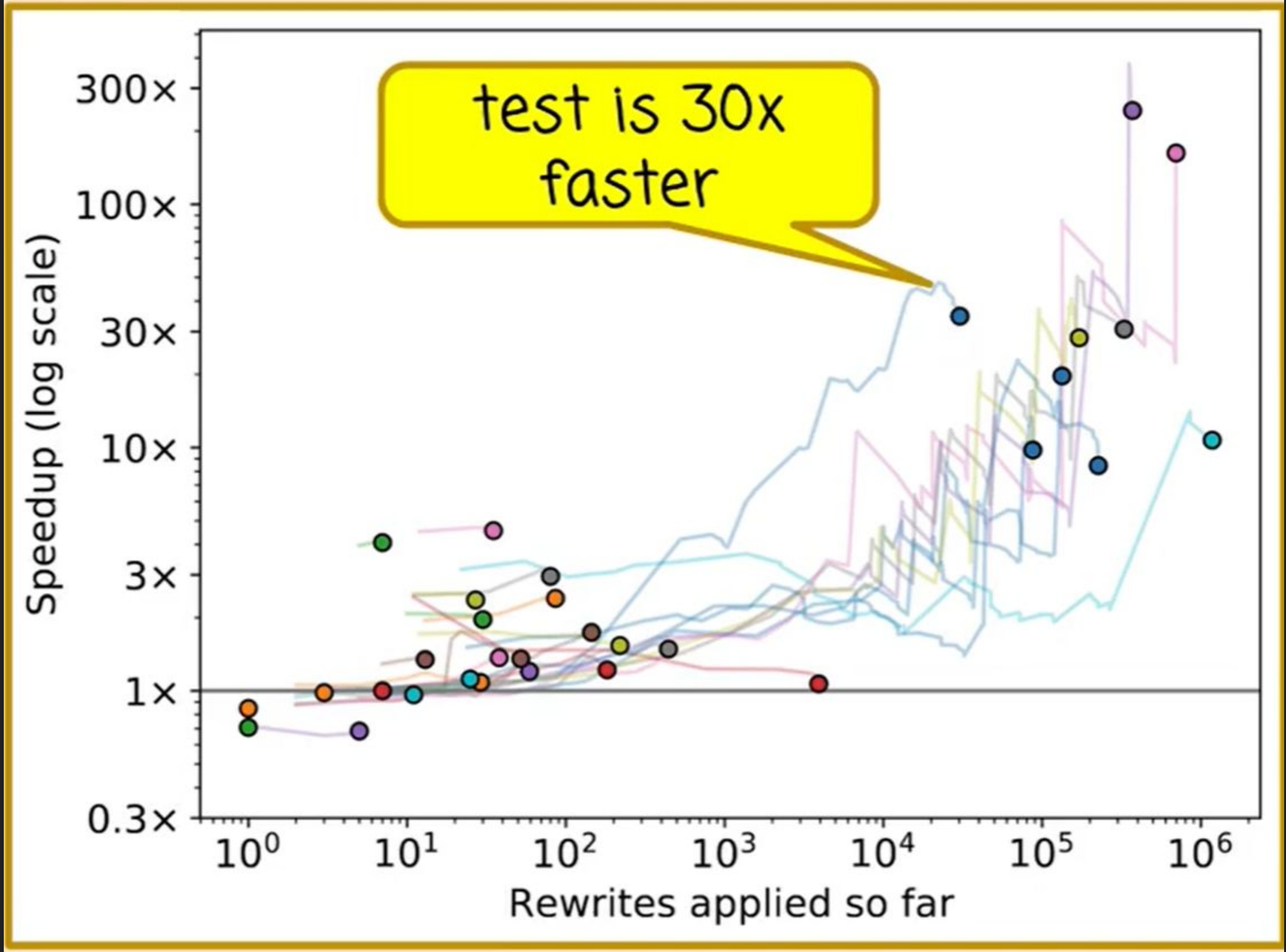
```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(expr)

  while not egraph.is_saturated_or_timeout():
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        ec2 = egraph.add(rw.rhs.subst(subst))
        egraph.merge(ec, ec2)


  return egraph.extract_best()
```

Efficient equality saturation



```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(expr)

  while not egraph.is_saturated_or_timeout():
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        ec2 = egraph.add(rw.rhs.subst(subst))
        egraph.merge(ec, ec2)

  return egraph.extract_best()
```

```
def equality_saturation(expr, rewrites):
  egraph = initial_egraph(expr)

  while not egraph.is_saturated_or_timeout():
    matches = []
    for rw in rewrites:
      for (subst, ec) in egraph.ematch(rw.lhs):
        matches.append((rw, subst, ec))
    for (rw, subst, ec) in matches:
      ec2 = egraph.add(rw.rhs.subst(subst))
      egraph.merge(ec, ec2)
    egraph.rebuild()

  return egraph.extract_best()
```


test is 30x faster
Speedup (log scale)
300×
100×
30×
10×
3×
1×
0.3×
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Rewrites applied so far

What about semantics?

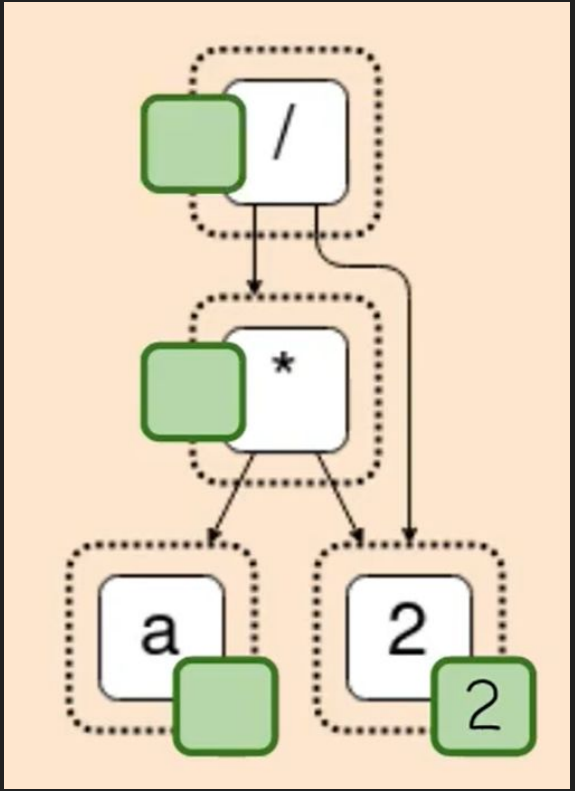
$$a \;/\; a \rightleftarrows 1$$

What about semantics?

$$a \;/\; a \rightleftarrows 1$$

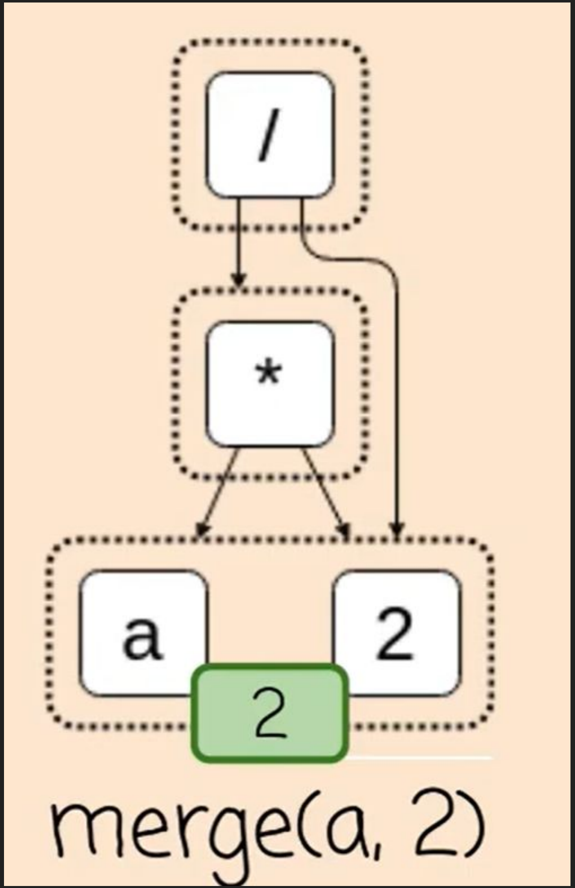
Constant folding

Analysis adds extra data to each e-class



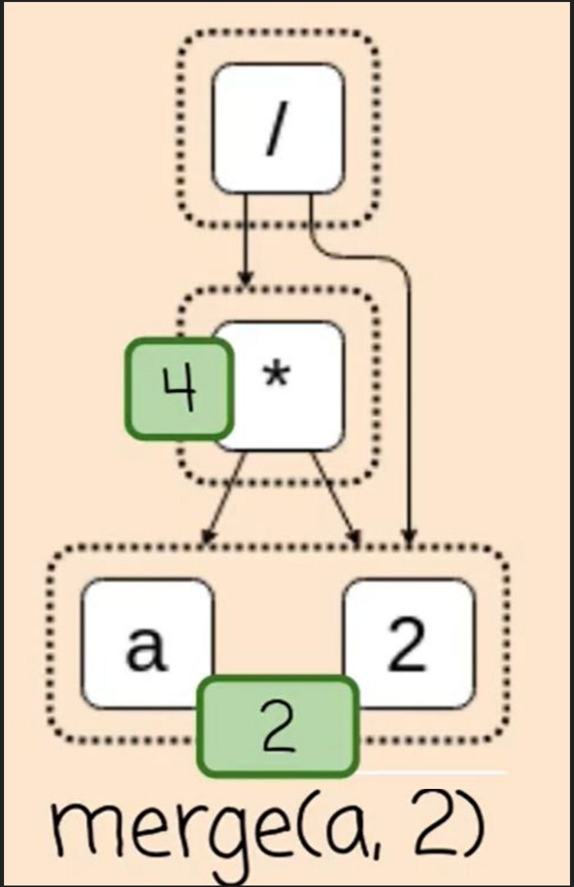
Constant folding

Analysis adds extra data to each e-class



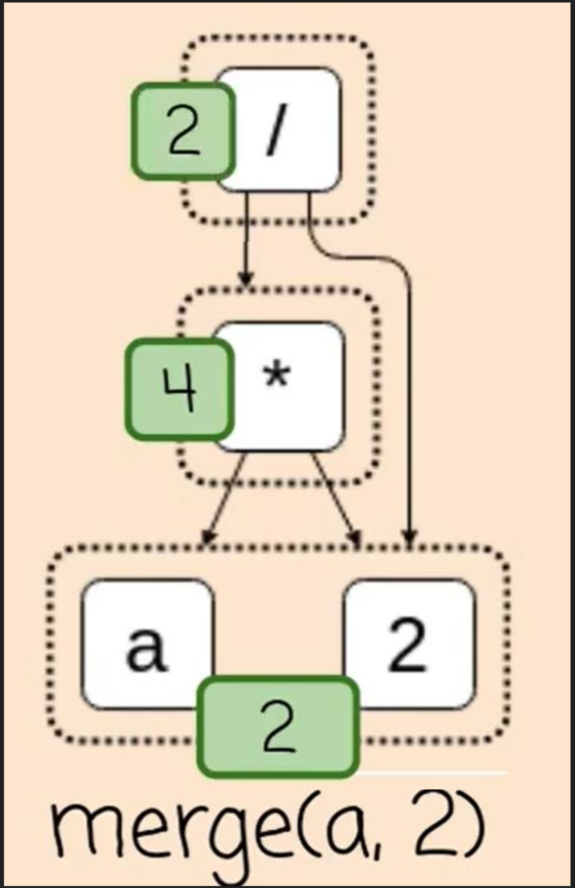
Constant folding

Analysis adds extra data to each e-class



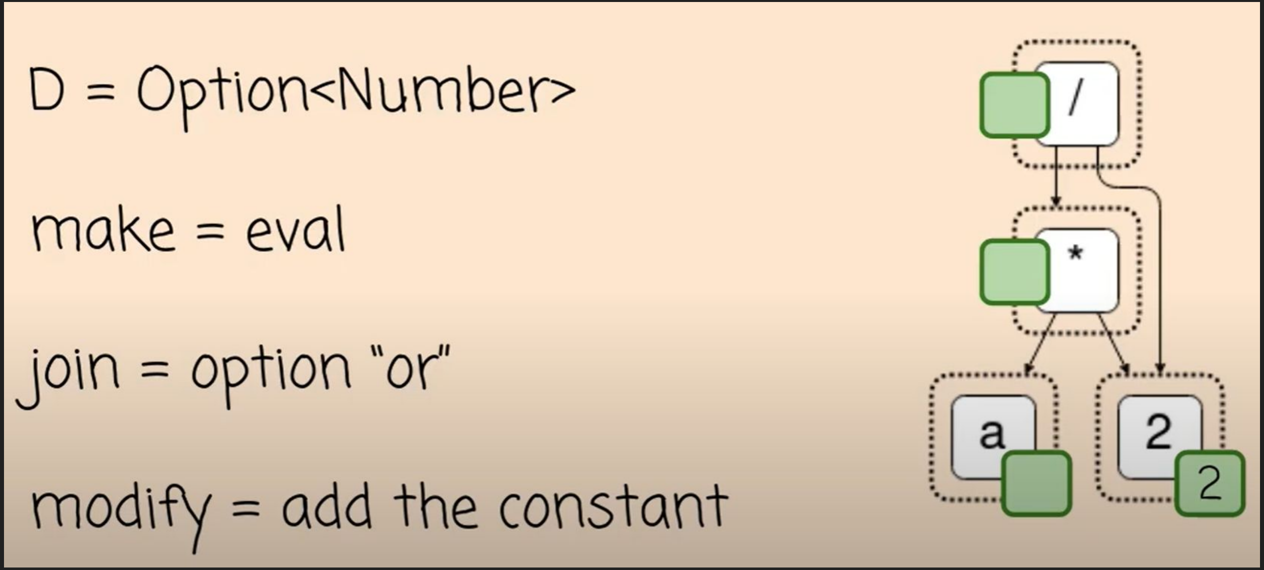
Constant folding

Analysis adds extra data to each e-class



All you need is ~~love~~ a lattice!

A join-semilattice (partial order with a least upper bound) will do.



D = Option<Number>

make = eval

join = option "or"

modify = add the constant

Analysis + E-classes = ❤

- Lift program analyses to e-class level
- Conditional & dynamic rewrites
- Other e-graph “hacks”
  - Pruning, debug assertions, on-the-fly extraction

The e-class analysis invariant

for each e-class

fixed point

$$\forall c \in G. \quad d_c = \bigwedge_{n \in c} \mathrm{make}(n) \quad \text{and} \quad \mathrm{modify}(c) = c$$

Analysis data is LUB
(lattice properties)

Other egg stuff

- Custom rewrites
- Logging
- Rule scheduling
- Batch simplification
- Saturation checking

Is it any good?

- Ruler automatically infers rewrite rules using equality saturation. OOPSLA 2021
- Diospyros automatically vectorizes digital signal processing code. ASPLOS 2021
- Tensat optimizes deep learning compute graphs both better and faster (up to 50x) than the state of the art. MLSys 2021
- Herbie improves the accuracy of floating point expressions. The egg-herbie library made parts of Herbie *over 3000x faster!* PLDI 2015
- Szalinski shrinks 3D CAD programs to make them more editable. PLDI 2020
- SPORES optimized linear algebra expressions up to 5x better than state-of-the-art. VLDB 2020
- Glenside explores the design space of hardware accelerators for a given deep learning program. MAPS 2021
- The folks at Intel have built a tool for Automating Constraint-Aware Datapath Optimization using egg. DAC 2023

Do you need to TURN ONE EXPRESSION INTO ANOTHER? USE EGG!

- E-graphs are efficient and general
- Avoid many headaches associated with term rewriting
- IR design is crucial

TITLE