Introduction to E-Graphs

Rebecca Swords | Women in Compilers and Tools

Questions

What are e-graphs?

What are they good for?

How do they work?

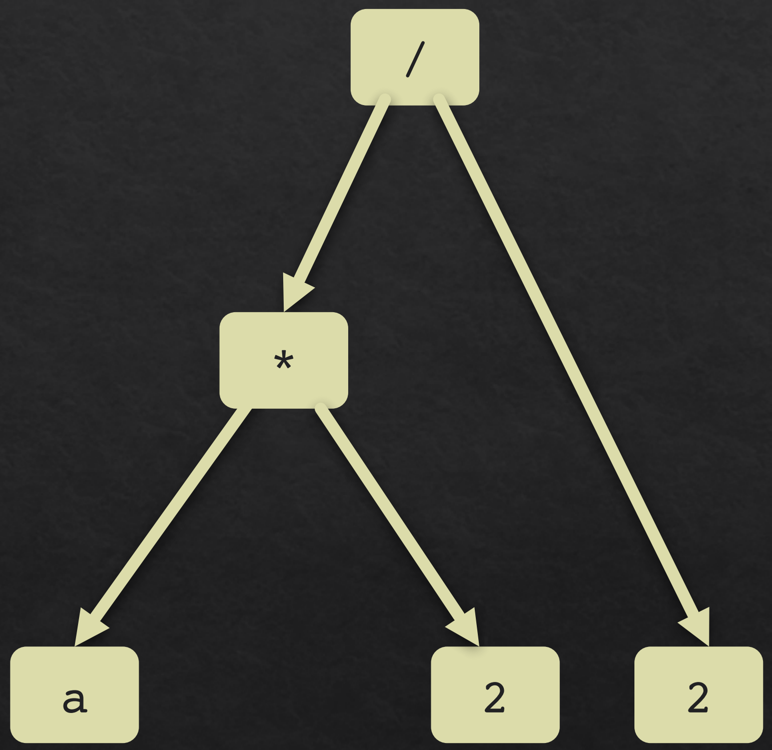
E-Graph

A data structure representing an equivalence relation over terms

Practical Applications

- Theorem proving
- SMT solving
- Optimization
- Translation validation
- Compilation
- Synthesis

Running Example: `(a*2)/2`



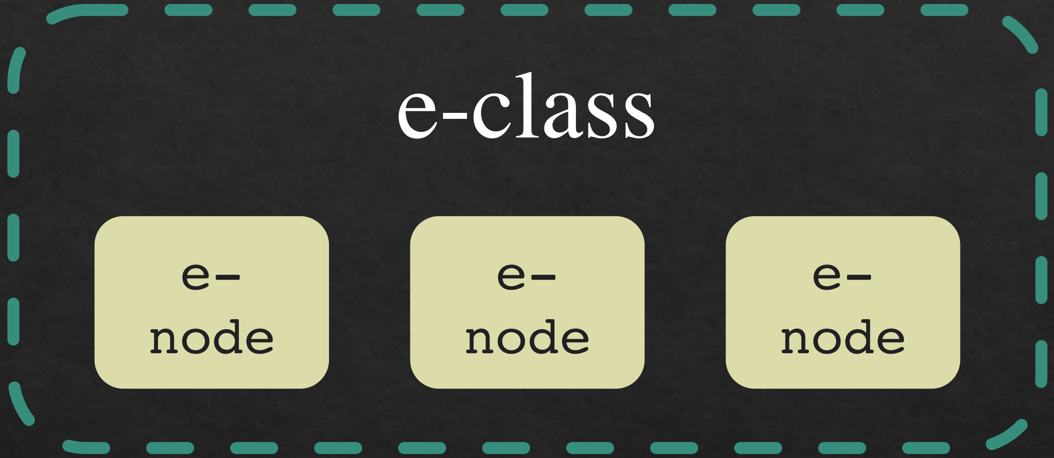
→ This reduces to `a`

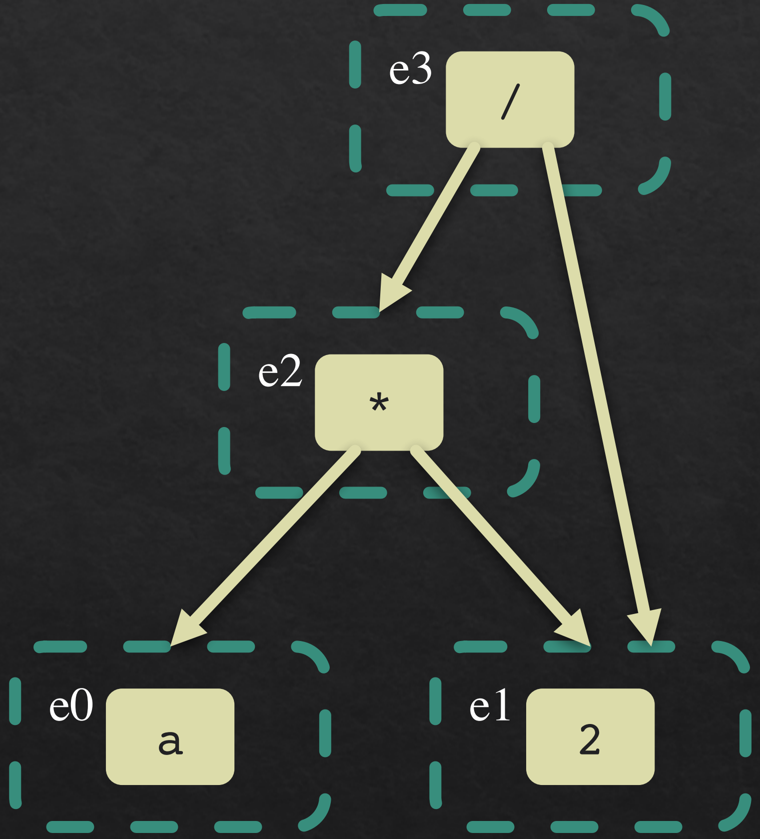
→ We can use e-graphs to do it!

Start with this AST

Into an E-Graph

Into an E-Graph





Build it with Quiche

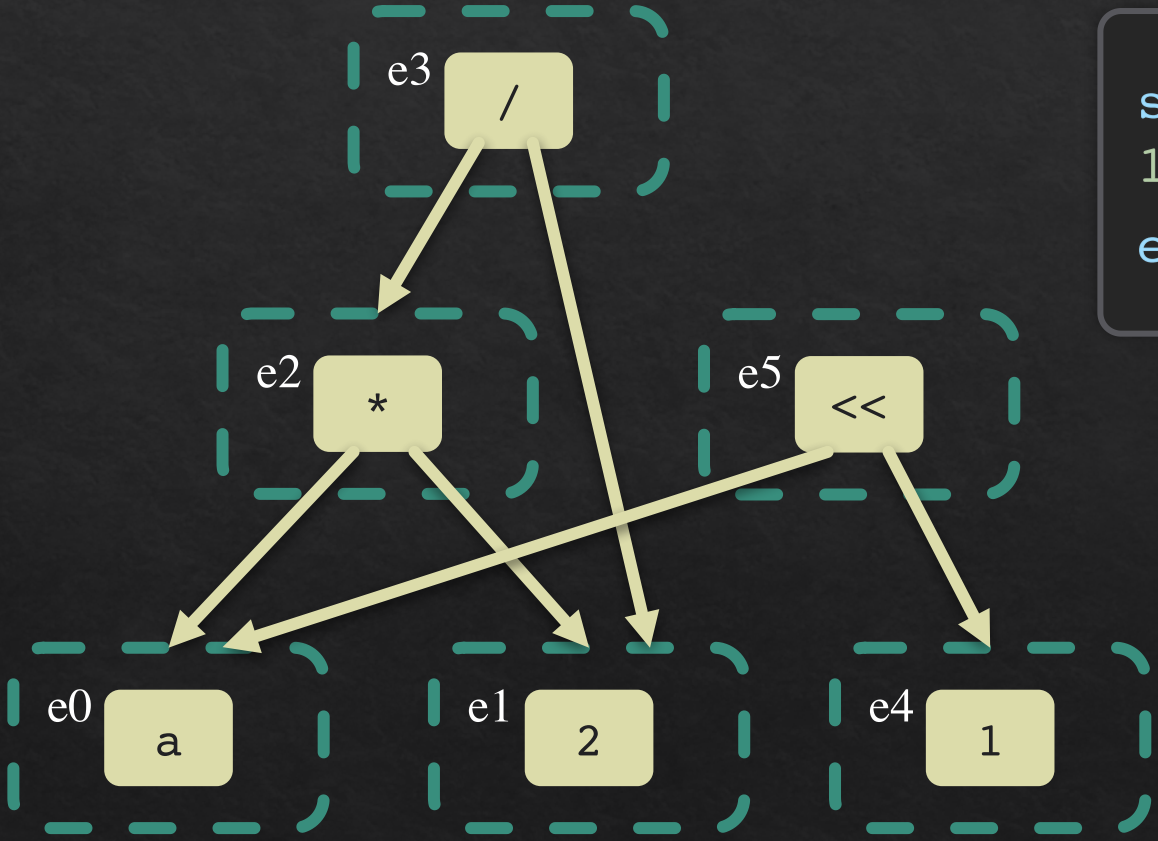
```
1 expr = (ExprNode('a', ()) * 2) / 2

2 quiche_tree = ExprTree(expr)

3 egraph = EGraph(quiche_tree)
```

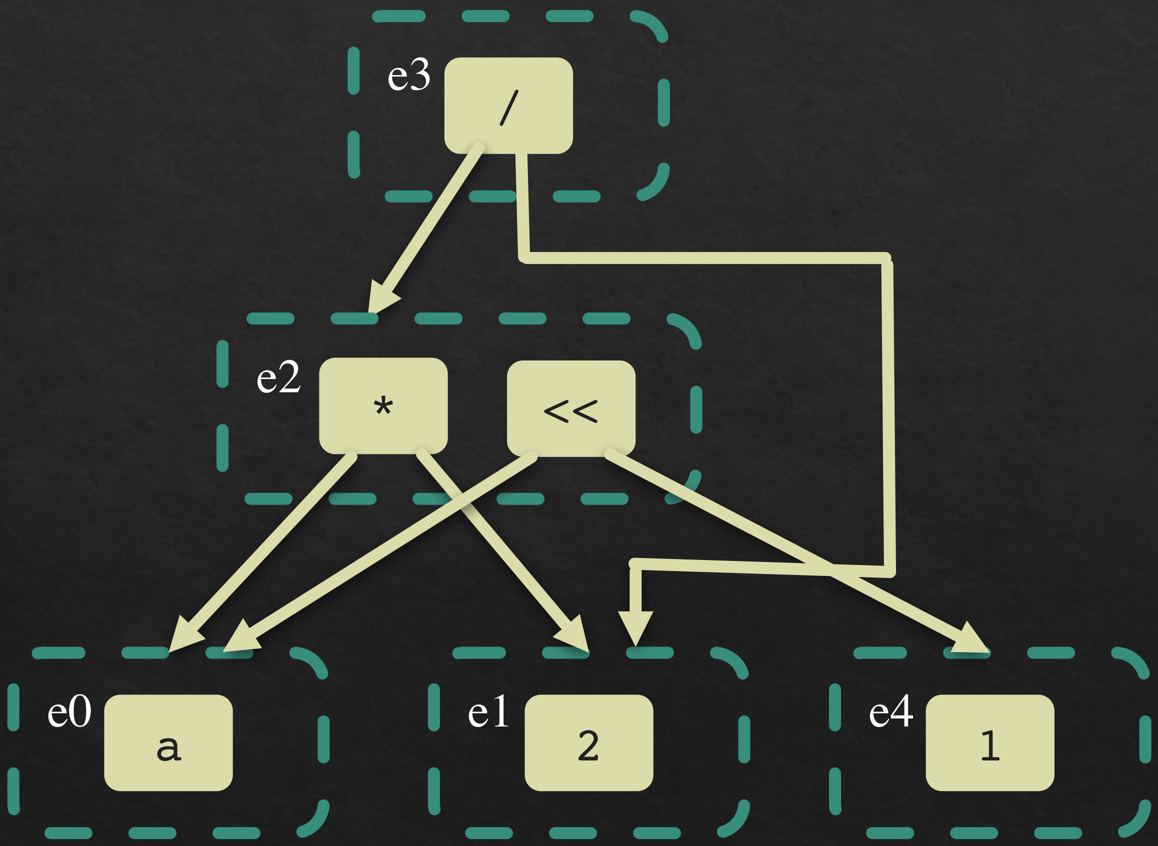
1. Parse term into the arithmetic language structure
2. Construct intermediate QuicheTree representation
3. Create e-graph from QuicheTree

Another Term: a << 1




```
shift_expr = ExprNode('a', ()) <<
1
egraph.add(ExprTree(shift_expr))
```

Merging Equivalent Terms



We assert:

a*2 === a<<1

It follows that:

(a*2)/2 === (a<<1) / 2

Manual Merging in Quiche

1
```
shift_eclass =
egraph.add(ExprTree(shift_expr))
times_node = ExprNode('a', ()) * 2
times_eclass =
egraph.add(ExprTree(times_node))
```

2
```
egraph.merge(times_eclass, shift_eclass)
```

3
```
egraph.rebuild()
```

1. Save e-class IDs for the expressions to be merged
2. Merge the two e-classes together
3. Restore e-graph invariants

E-Graphs More Formally

Structure

- E-node: an n-ary function symbol and n children (e-class IDs)
- E-class: set of e-nodes
- Union-find over e-classes: `add`, `merge`, `find` operations
- Canonical e-node: for each child, i, `find(i) = i`
- Hashcons: maps canonical e-nodes to e-classes

Invariants

- Hashcons maps all canonical e-nodes
- Equivalence closed under congruence, i.e., congruent e-nodes are in the same e-class

```
If a = b, then f(a) = f(b)
```

Why is this good for term rewriting?

Instead of destructive rewrites, put *all* equivalent terms in the e-graph

- → No worries about phase ordering
- → Consider all options and choose the “best” at the end

E-Matching

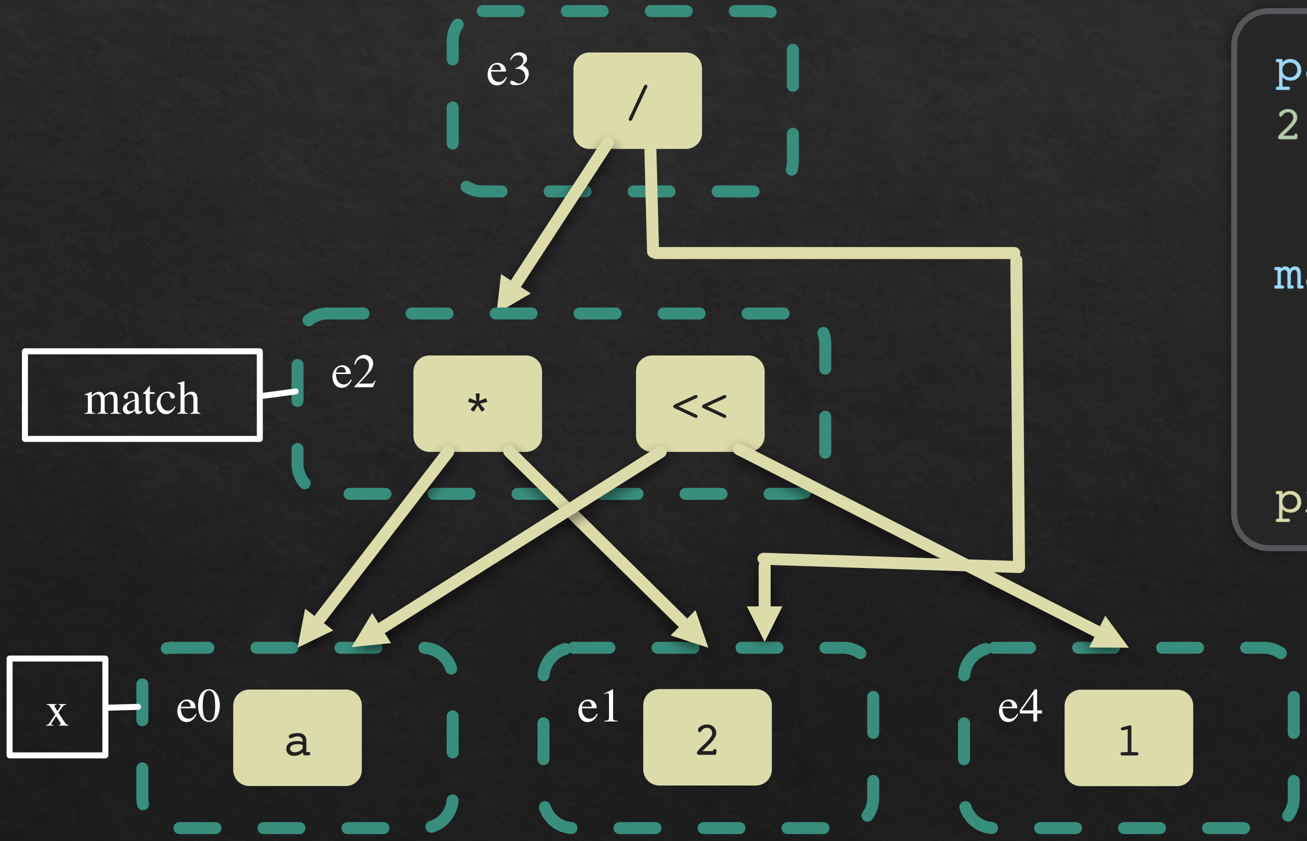
Pattern matching for e-graphs!

→ Add pattern variables to language

→ `ematch` searches for a pattern and returns:

- e-class matching the term
- substitution from vars to e-class IDs

E-Matching Example: x * 2




```
pattern = ExprTree(ExprNode('x', ()) *
2)

matches = egraph.ematch(pattern,
    egraph.eclasses())

print(matches)
```

```
[ (e2, { 'x': e0 }) ]
```

Rewriting Rules: Pattern Merges

1
```
rule = ExprTree.make_rule(lambda x:
    (x * 2, x << 1))
```

2
```
Rule.apply_rules([rule], egraph)
```

3
```
print("Shift e-class: ", shift_eclass)
print("Shift e-class.find(): ",
shift_eclass.find())
```

1. Create a rule:

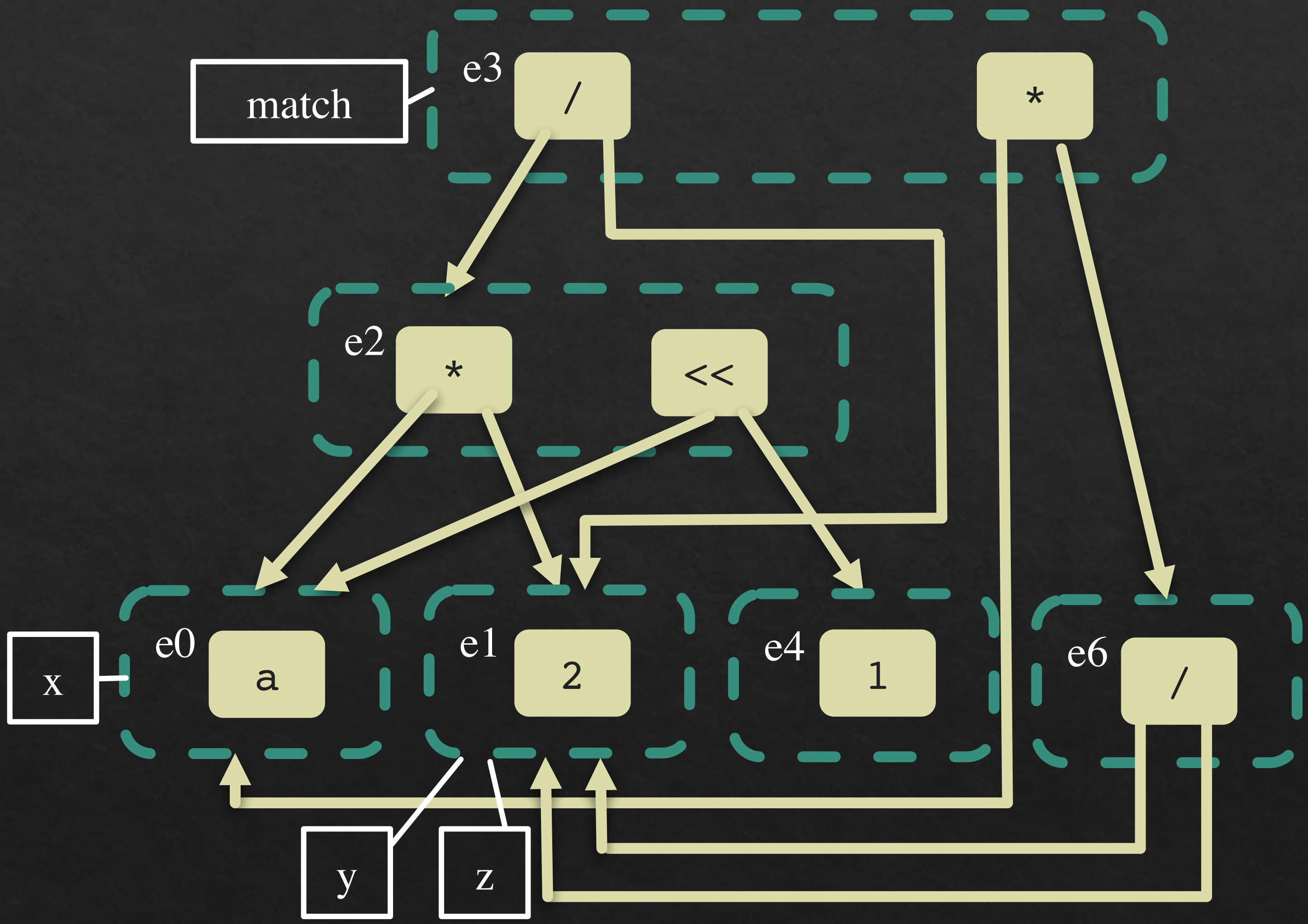
   `x * 2 === x << 1`

2. Apply all rules to e-graph (and rebuild)

3. `Shift e-class: e5`
   `Shift e-class find: e2`

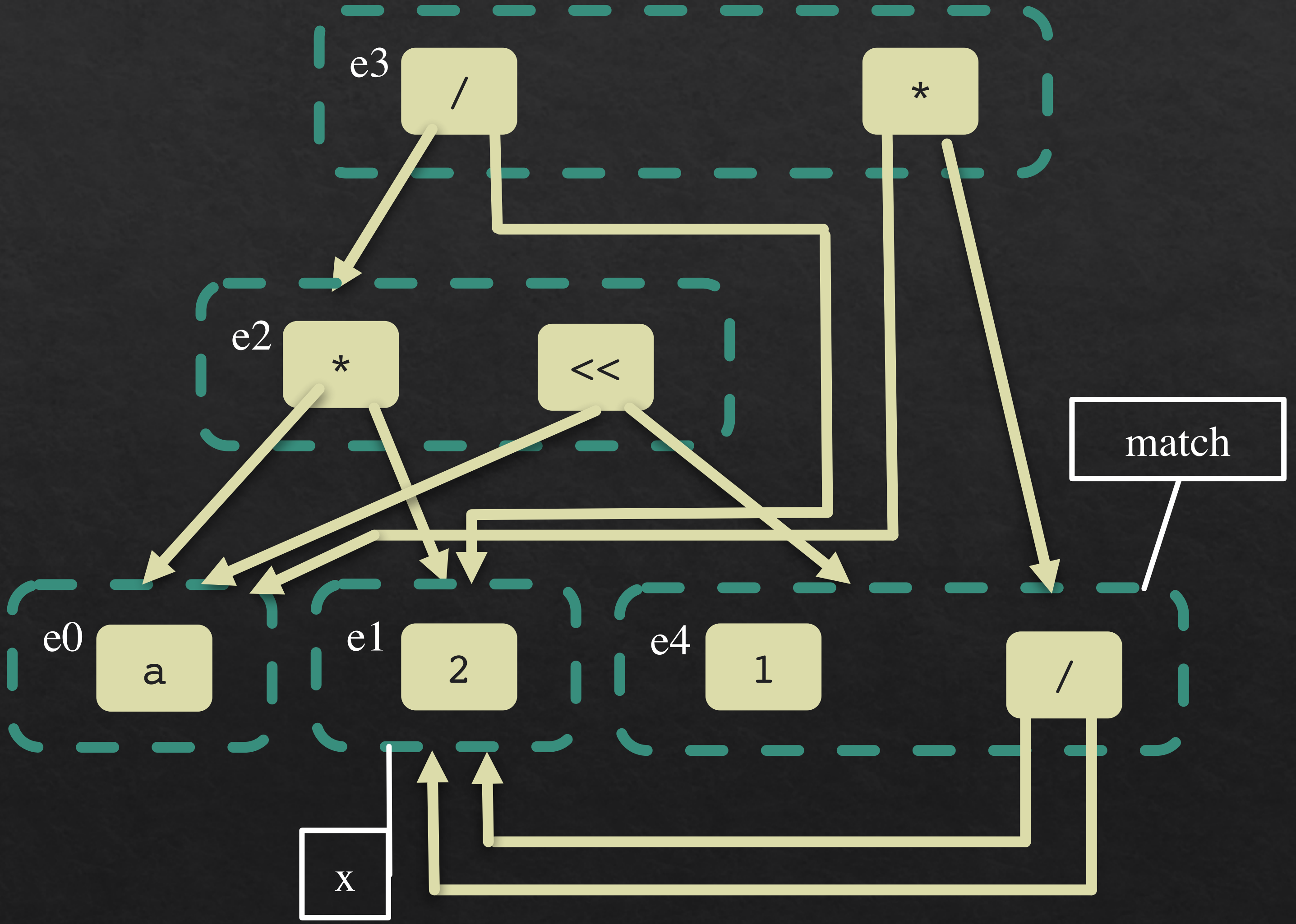
Another rewrite:

```
(x*y)/z
===
x*(y/z)
```



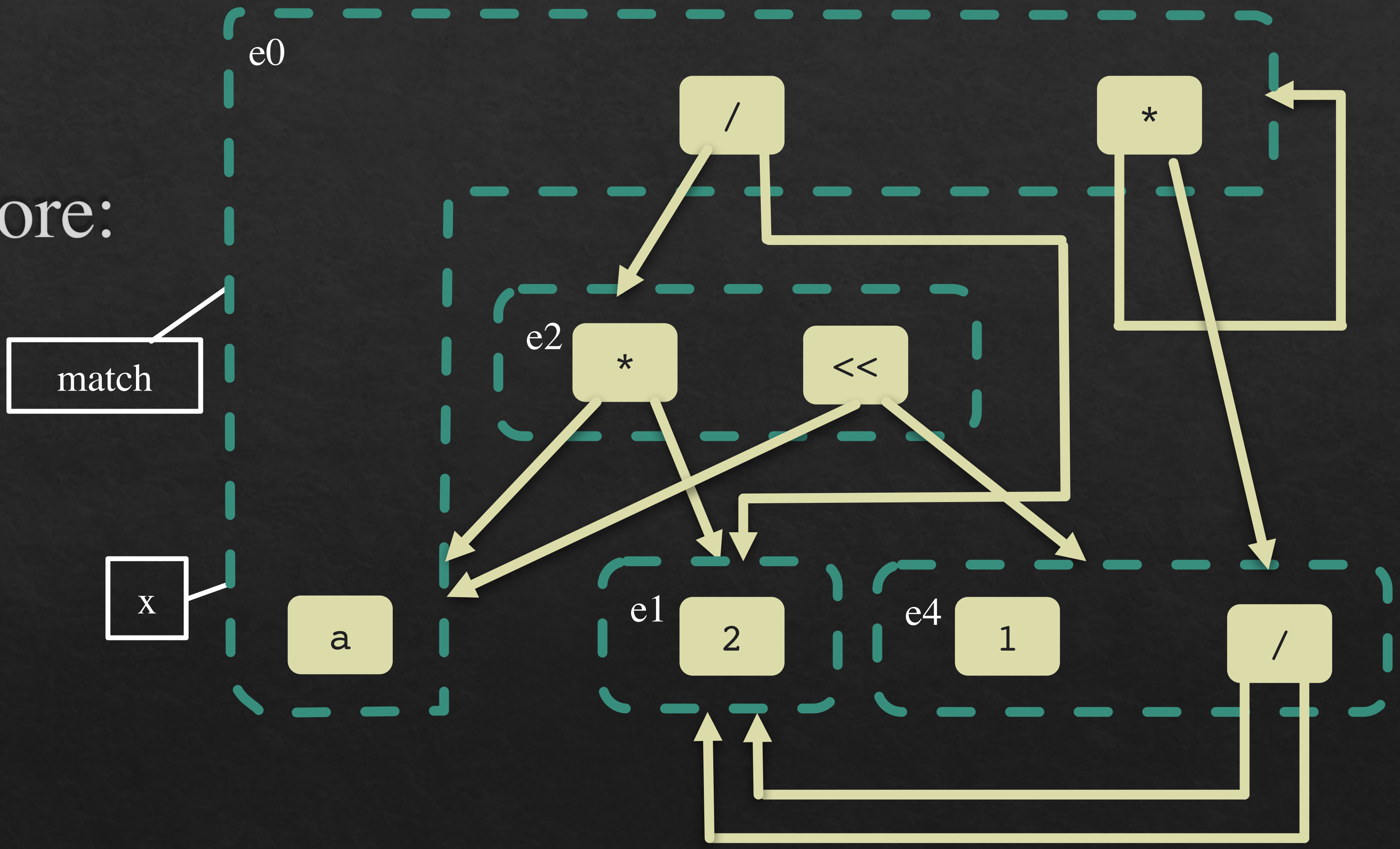
And another:

x/x
===
1



And one more:

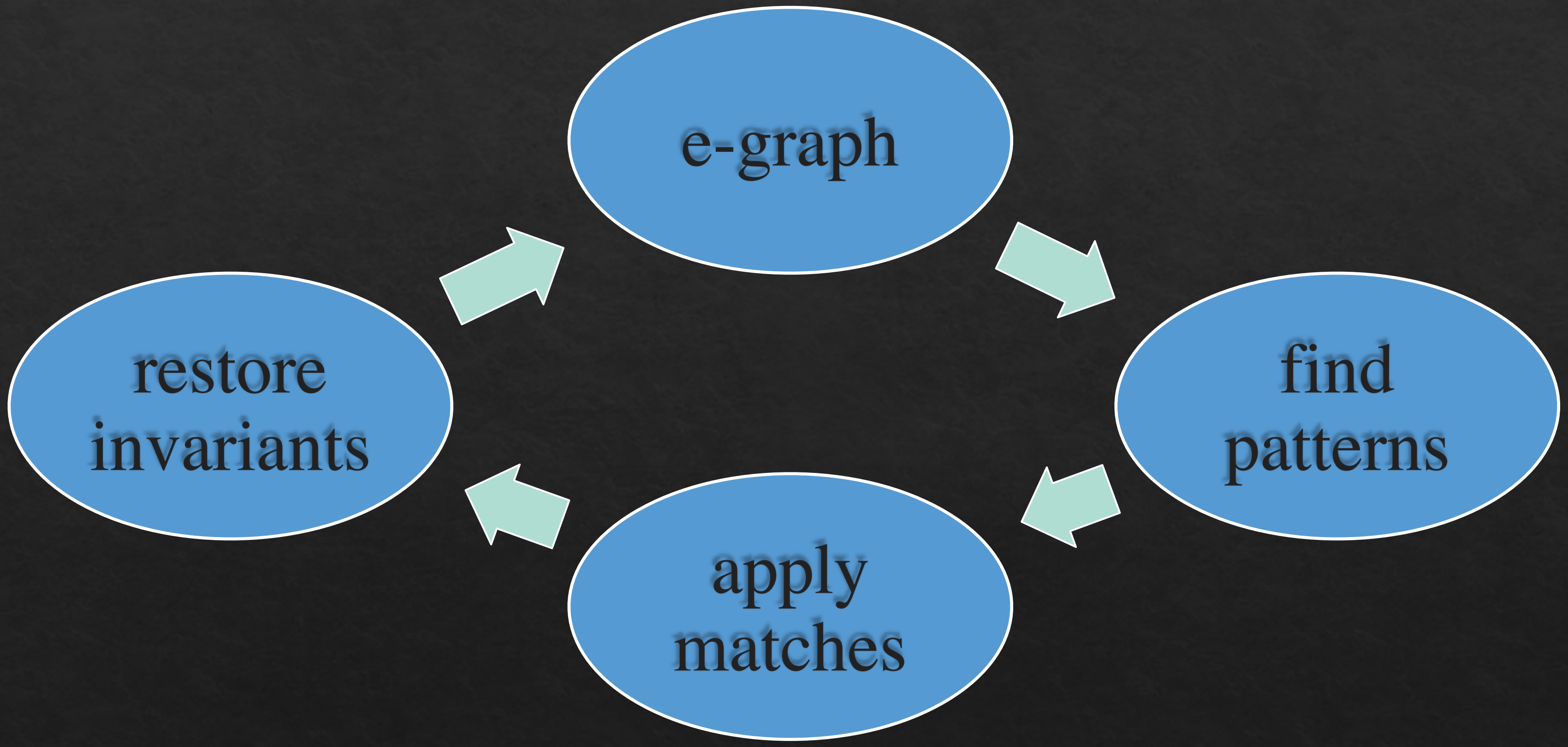
```
x*1
===
 x
```



Equality Saturation

Keep applying rewrite rules until no new changes are made

Equality Saturation Loop



Apply Rules Until Saturation

```
1 rules = [
      ExprTree.make_rule(lambda x, y, z:
          ((x * y) / z, x * (y / z))),
      ExprTree.make_rule(lambda x:
          (x / x, ExprNode(1, ()))),
      ExprTree.make_rule(lambda x: (x * 1, x))
  ]
  while not egraph.is_saturated():
2     Rule.apply_rules(rules, egraph)
  aeclass = egraph.add(ExprTree(ExprNode('a',
  ()))))
3 assert aeclass.find() == egraph.root.find()
```

1. Same 3 rules we just applied
2. Apply rules until the e-graph is saturated
3. Verification: expect `a` to have merged with the “root” e-class

E-Class Analysis

Domain-specific e-graph extensions

- Attach datum to each e-class based on e-nodes: `make`
- Merge data when e-classes merge: `join`
- Update e-class based on datum: `modify`
- Form a join-semilattice

What Can E-Class Analyses Do?

- Program analysis
- Conditional or dynamic rewrites
- Debugging
- Pruning
- On-the-fly term extraction

Standardized interface for extending e-graphs!

Analysis Invariant

for each e-class

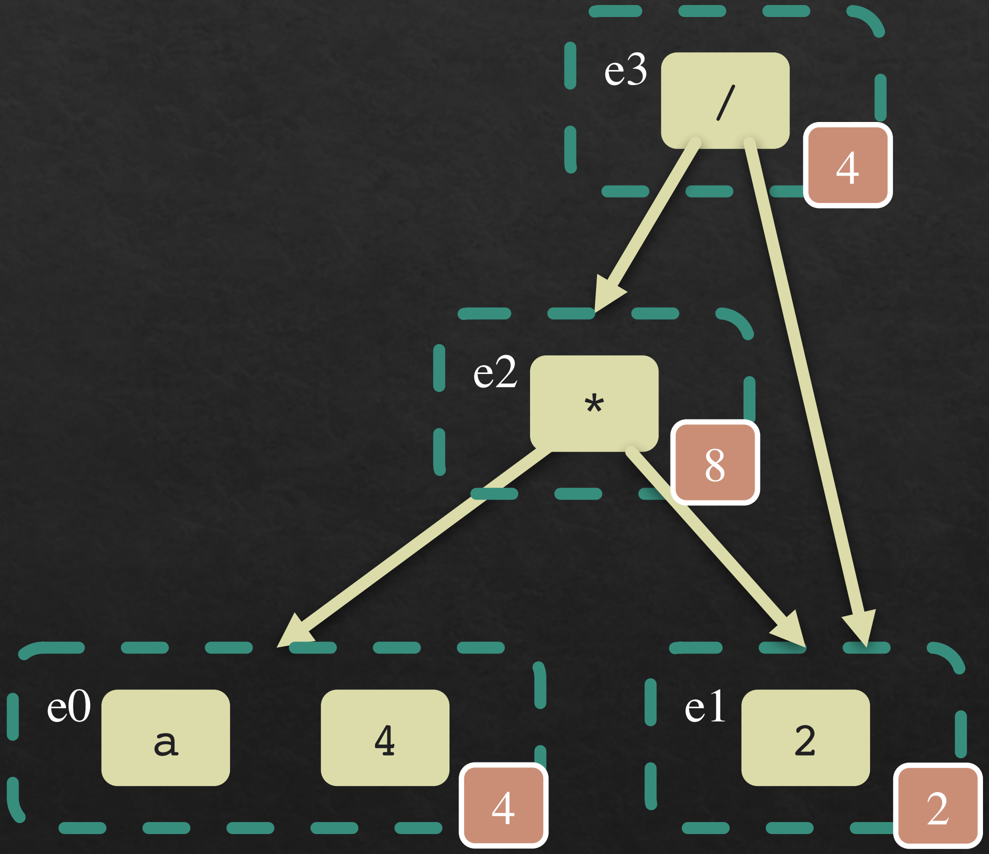
fixed point

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

data is the same as \`make\`-ing data for each e-node and then \`join\`-ing

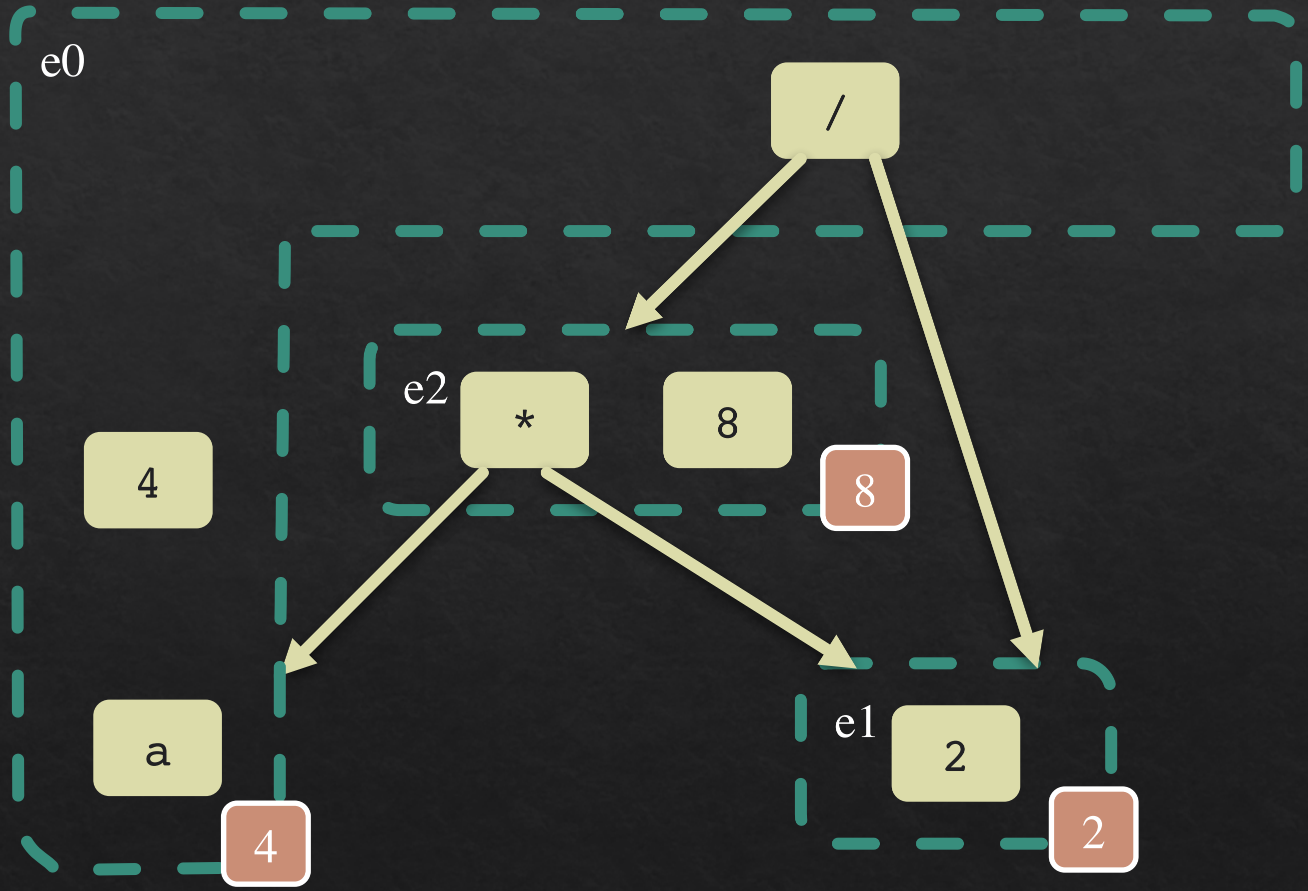
Constant Folding E-Class Analysis

Suppose we learn that a === 4



Constant Folding
E-Class Analysis

Suppose we learn that a === 4



Constant Folding: Usage

```
1 expr = (ExprNode('a', ()) * 2) / 2
  quiche_tree = ExprTree(expr)
  egraph = EGraph(quiche_tree, ExprConstantFolding())
2 four_eclass = egraph.add(ExprTree(ExprNode(4, ())))
  a_eclass = egraph.add(ExprTree(ExprNode("a", ())))
3 egraph.merge(a_eclass, four_eclass)
4 egraph.rebuild()
5 assert egraph.root.data == 4
```

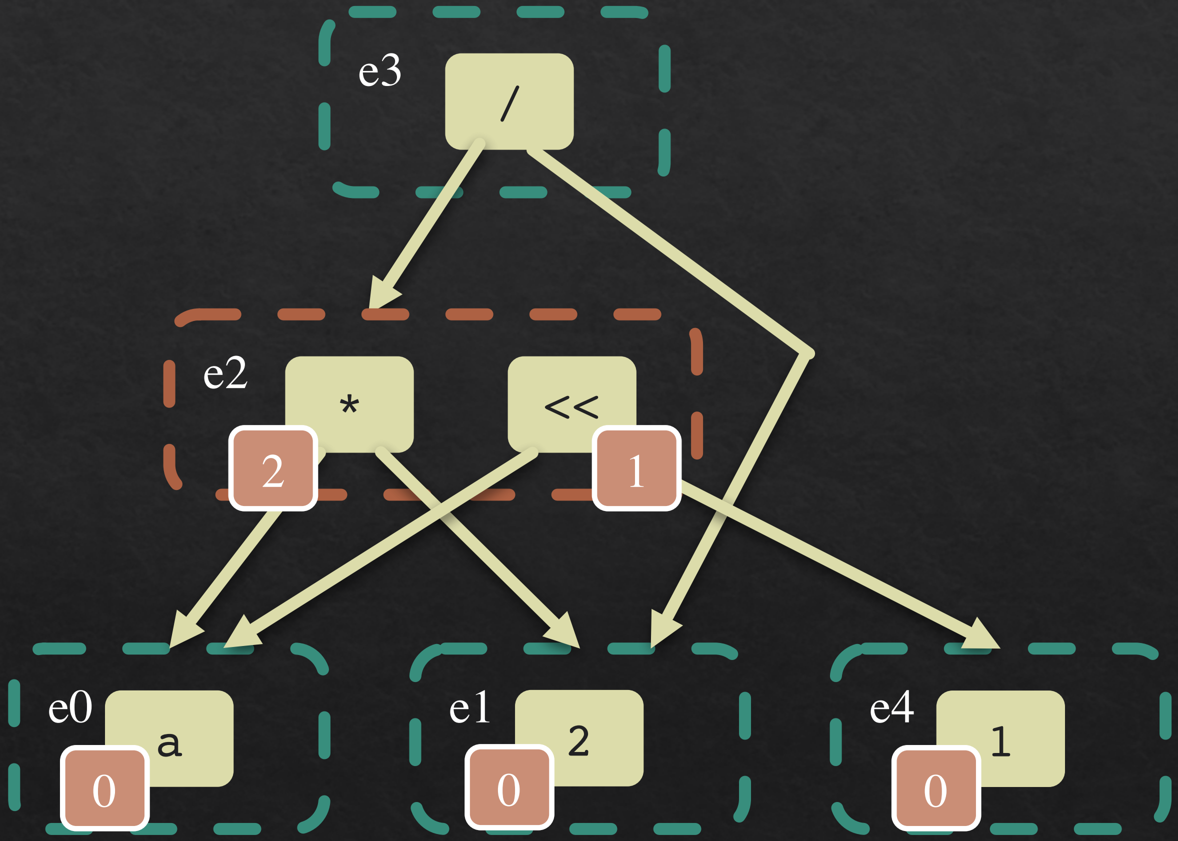
1. Create e-graph with constant folding analysis
2. Get e-class IDs
3. Merge 4 with a
4. Rebuild (update analysis)
5. Verify

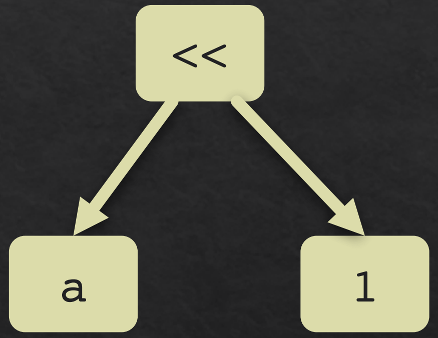
Term Extraction

- Pick an e-class to extract
- Cost model assigns a cost to e-nodes
- Choose best e-node for each e-class
- Construct a term by combining the e-node values

Term Extraction Example

Operator	Cost
+	1
<<	1
*	2
/	3
default	0





Term Extraction Example

```
1 cost_model = ExprNodeCost()
  extractor= MinimumCostExtractor()
2 extracted = extractor.extract(
      cost_model,
      egraph,
3     egraph.root.find(),
4     ExprTree.make_node)
  assert str(extracted) == "a"
```

1. Initialize cost model and extractor
2. Extract the best term
3. Specify which e-class to extract
4. Function to construct ExprTree from e-node data

More on Quiche

- Add your own languages!
  - Bring your own parser, adapt your AST into a QuicheTree
- End-to-end Python rewriting!
  - Uses native Python parser (v3.7+)
  - Read/write valid Python files
- Native Python!
  - With all its pros and cons

QuicheTree

Quiche requires the user to provide a parsed tree that implements `QuicheTree` (“*bring your own parser*”).

`value()`
the e-node key

`children()`
list of the node's children

`is_pattern_symbol()`
for e-matching; indicates if the node is a pattern

```
class QuicheTree(ABC):
  @abstractmethod
  def value(self)

  @abstractmethod
  def children(self)

  @abstractmethod
  def is_pattern_symbol(self)
```

Links and References

- Quiche repo: https://github.com/riswords/quiche
- egg website: https://egraphs-good.github.io/
- egg: Fast and extensible equality saturation (POPL '21, Willsey, et al.): https://dl.acm.org/doi/10.1145/3434304
- Equality-Based Translation Validator for LLVM (CAV '11, Stepp, Tate, & Lerner): https://cseweb.ucsd.edu/~rtate/publications/eqsat/eqsat_stepp_cav11.pdf
- babble: Learning Better Abstractions with E-Graphs and Anti-Unification (POPL '23, Cao, et al.): https://dl.acm.org/doi/10.1145/3571207

Questions?

Additional References from Q&A

1. Link to the public E-Graphs Zulip chat: https://egraphs.zulipchat.com/
2. Perfect Reconstructability of Control Flow from Demand Dependence Graphs (Bahmann, et al. 2014) https://dl.acm.org/doi/abs/10.1145/2693261
3. E-Graphs Zulip discussion of using RVSDG representation: https://egraphs.zulipchat.com/#narrow/stream/328976-Program-Optimization/topic/PEGs
4. Equality Saturation for Tensor Graph Superoptimization (Yang, et al., MLSys 2014): https://arxiv.org/abs/2101.01332
5. Relational e-matching (Zhang, et al., POPL 2022)
6. Logging an Egg: Datalog on E-Graphs (EGRAPHS 2022) - PLDI 2022 (sigplan.org)