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: \( \left( a \times 2 \right) / 2 \)

- This reduces to \( a \)
- We can use e-graphs to do it!

Start with this AST
Into an E-Graph
Into an E-Graph

e-class

e-node  e-node  e-node

e0  e1  2

e3

e2

/
Build it with Quiche

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

expr = (ExprNode('a', ()) * 2) / 2
quiche_tree = ExprTree(expr)
egraph = EGraph(quiche_tree)
Another Term: a << 1

```python
code
shift_expr = ExprNode('a', ()) << 1
egraph.add(ExprTree(shift_expr))
```
We assert:
\[ a \times 2 \equiv a \ll 1 \]

It follows that:
\[ (a \times 2) / 2 \equiv (a \ll 1) / 2 \]
Manual Merging in Quiche

1. Save e-class IDs for the expressions to be merged

```python
def shift_expr():
    return 'a'

times_node = ExprNode('a', ()) * 2

times_eclazz =
    egraph.add(ExprTree(times_node))

egraph.merge(times_eclazz, shift_eclazz)
```

2. Merge the two e-classes together

3. Restore e-graph invariants

```python
egraph.rebuild()
```
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, \( \text{find}(i) = i \)
- 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: $x \times 2$

```
pattern = ExprTree(ExprNode('x', ()) * 2)
matches = egraph.ematch(pattern, egraph.eclASSES())
print(matches)
```

```
[(e2, {'x': e0})]
```
Rewriting Rules: Pattern Merges

1. Create a rule:
   \[ x \times 2 \Rightarrow x \ll 1 \]

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:

\[
x/x = (x^0) = 1
\]
And one more:

\[ x \cdot 1 \quad \text{match} \]

\[ === \]

\[ x \]

\[ x \]
Keep applying rewrite rules until no new changes are made
Equality Saturation Loop

e-graph

find patterns

apply matches

restore invariants
Apply Rules Until Saturation

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

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

- for each e-class
- fixed point
- 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 $a \equiv 4$
Constant Folding: Usage

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

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>3</td>
</tr>
<tr>
<td>default</td>
<td>0</td>
</tr>
</tbody>
</table>
Term Extraction Example

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

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

```python
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](https://github.com/riswords/quiche)
- egg website: [https://egraphs-good.github.io/](https://egraphs-good.github.io/)
Questions?
Additional References from Q&A

1. Link to the public E-Graphs Zulip chat: https://egraphs.zulipchat.com/
5. Relational e-matching (Zhang, et al., POPL 2022)
6. Logging an Egg: Datalog on E-Graphs (EGRAPHS 2022) - PLDI 2022 (sigplan.org)