`egg`: Fast and Extensible Equality Saturation - Summary

Concepts

- E-graphs - graph structure used to represent programms that allows for equality saturation
- E-class - all expressions that are equivalent belong to an e-class, each e-node starts in its own e-class
- E-node - represents an expression, along with its subexpressions

Equality Saturation

1. Start by translating our expression (programm) into an e-graph representation
2. Define a set of equality rules for expression matching that will be used to extend (saturate) the e-graph
3. Egg maintains a UnionFind like structure to remember which e-nodes belong to which e-classes
   - For example expressions (e-nodes) `2 * x` and `x + x` are equivalent and thus would be merged into a single e-class
4. Merging nodes is done by pattern matching against the rules. First the matches are found and then their e-classes are merged together.
5. After 2 e-classes get merged it may happen that the "parent" expression becomes equivalent
   - For exmaple we have `e_1 = 2 * x_1` and `e_2 = 2 * x_2`; then we discover that `x_1 = x_2`; therefore we also need to merge the parent expressions `e_1` and `e_2`; this upwards merging should be applied recursively to parent expressions
6. After fully saturating the graph by applying the rules (or a timeout), the e-graph contains our original expression, as well as the derived equivalent ones
7. Lastly we can extract an optimal (according to some cost function) expression that is equivalent to the original expression

`egg` nuances

- The upwards merging operation is "computationally expensive". In order to mitigate this `egg` saturates the e-graph in iterations. Each iteration consists of finding matches in the e-graph, then applying the merges to matched e-nodes, and lastly fix the e-graph invariant - the merging of other e-nodes that got implicitly merged.
- The equality saturation is easily extensible by e-class analysis to allow optimisations like constant folding to be an inherent part of equality saturation

- The rewrites of the e-graph are truly unordered by separating the expression matching phase from the rewrite (merging phase). This way rules that are later are not prioritised by having more opportunities to match.
- Rules that are matched very often are scheduled to pattern match more rarely. This is a heuristic that often leads to faster equality saturation or to more meaningful and useful e-graph if the saturation time-outs.