Your next e-graph framework looks like Datalog

Yihong Zhang
University of Washington
USA
yz489@cs.washington.edu

Recent years have seen a rejuvenation of e-graphs in many areas, ranging from floating-point arithmetic [14] to machine learning compiler [16] and from computational fabrication [11] to rule synthesis [12]. Several e-graph frameworks emerge as a result [2, 5, 15]. Among these frameworks, egg, the e-graph framework that first introduced the idea of rebuilding and e-class analyses, greatly expands the capability of the e-graph data structure by scaling up congruence maintenance with novel algorithms and by providing an extensible interface based on e-class analyses.

However, the wide variety of e-graph applications is placing new requirements on the capability of e-graph frameworks. Some of these requirements are difficult to fulfill in existing frameworks like egg. For example, applications like e-graph–based tensor graph optimizations [16] use a standard extension to e-matching called multi-patterns, which egg does not readily support (as of mid-Mar 2022). A pull request was made recently in an attempt to support multi-patterns. However, it requires modifications to the current rewrite interface, and the proposed implementation does not benefit from recent progress in e-matching [17]. As another example, while equational reasoning is efficiently supported in egg, non-symmetric reasoning like the logical implication relation is fairly non-trivial and potentially inefficient in egg. Yet practical reasoning may interleave symmetric relations with non-symmetric ones. To remedy these deficiencies, we need a radical change to the current interface. We call the new interface design egg 2.0.

What could egg 2.0 look like? Advances in e-graph and database researches give us some clues. In previous work [17], we improved e-matching by reducing it to queries over relational databases. This hints at the underlying connection between e-graphs and relational databases. Moreover, Datalog, a fixpoint-based relational language, is able to express various non-symmetric relations (e.g., reachability) and can compute them using efficient algorithms (e.g., the semi-naïve evaluation algorithm [6]). Modern Datalog engines are also being extended to support efficient equational reasoning. For example, Soufflé has first-class support for efficient equivalence relations [13]. Finally, the semiring [7, 9] and lattice [3, 10] semantics of relational databases precisely capture the monotonic nature of e-class analyses.

We argue that egg 2.0 should be a Datalog language. This will at least have the following advantages.

1. A relational representation of e-graphs in Datalog could improve the performance of e-matching asymptotically in many cases, as shown in our previous work on relational e-matching.
2. The efficient evaluation algorithms designed for Datalog, such as semi-naïve evaluations, could benefit rule rewriting in e-graphs.
3. Rules in Datalog are naturally multi-patterns. This will allow first-class support for multi-patterns, whose performance will also benefit from relational e-matching.
4. The well-studied lattice semantics of relations supports and in fact generalizes e-class analyses in egg.

As our first step, we built egg💡[1], a relational e-graph framework on top of SQLite. In previous work, we build a prototype implementation for e-graphs with SQLite as well [17]. However, that prototype only supports basic e-graph operations like insertion and merging. egg💡 significantly expands its usability with support for match-apply iterations and multi-patterns, which allows users to write real-world applications like equality saturation. To our knowledge, this is the first full-fledged e-graph framework fully on top of a relational database.

egg💡 can be viewed both as a multi-pattern equational reasoning language for e-graphs and as a Datalog language with an internalized notion of congruence. egg💡 has a Datalog-like surface language, and it translates e-graph operations into SQL statements, which are executed in SQLite. In egg💡, we use relation $\texttt{Add}(x_1, x_2, c)$ to represent a term $\texttt{Add}(x_1, x_2)$ with e-class id c. The associativity rule of Add can be specified as follows:

$$(\texttt{Add}\ \alpha_1\ (\texttt{Add}\ \alpha_2\ \alpha_3))@\alpha_4 \Rightarrow (\texttt{Add}\ (\texttt{Add}\ \alpha_1\ \alpha_2)\ \alpha_3)@\alpha_4.$$

The annotation syntax $p@\alpha$ means that α denotes the id of (sub)pattern p. The semantics of rules is as follows: for each pattern on the left-hand side, substitute and populate patterns on the right-hand side and unify the patterns annotated with the same id.

egg💡 is performant. It uses a novel algorithm for performing batched rewrites. Such an algorithm is reminiscent of the chase [1], a procedure used in relational databases for repairing databases using functional dependency. It also uses an efficient algorithm adapted from egg for maintaining congruence. Both algorithms lay a solid foundation for efficient computations in egg 2.0. egg💡 also demonstrates how far we

PL'18, January 01–03, 2018, New York, NY, USA
2018.

[1] Pronounced as egg lite.

egg	egg♯
Equational rewrites	Datalog rules
Congruence rules	Functional dependencies (FD)
E-classes	User-defined sorts
E-class merges	FD repair through unification
E-class analyses	User-defined lattices
E-class analysis maintenance	FD repair through lattice joins

Figure 1. The correspondence between constructs in egg and in egg♯.

can push the limits of an in-memory database system and use it as an e-graph engine. The preliminary benchmark shows that, even with the overheads for interpretation, parsing, and communication, egg♯ is within one order of magnitude slower than egg, which is highly customized for the e-graph workload.

egg♯ also exhibits some interesting designs. For example, different from a traditional e-graph implementation, egg♯ does not have a global union-find data structure. Instead, a local union-find is created transiently during each rebuilding. This challenges the traditional view that an e-graph is a DAG of terms and an equivalence relation over the terms [4]. In fact, the design of egg♯ demonstrates that the equivalence relation in an e-graph can be avoided with aggressive normalization. Moreover, although our current egg♯ implementation only supports congruence rules, we realized that congruence is just a special kind of functional dependency over the database. This again challenges the long-held belief that congruence is an essential ingredient to e-graphs, inspiring future designs for egg 2.0. We have discovered several interesting non-congruent functional dependencies so far. One of them is the destructor. For example, snoc, the dual to cons, takes a list l and returns x and l', the head and tail of the list. In egg♯, it is represented as $\mathsf{snoc}(l, x, l')$ and this multi-output operator naturally has a functional dependency $l \longrightarrow x, l'$, meaning that each l uniquely determines both x and l'. Unification (with injective rules) is another kind of functional dependency that we find interesting. For example, the unification closure of a type theory with function type $\rightarrow$ and type variables will have the following rule (besides transitivity, reflexivity, and commutativity of $\equiv$):

$$\frac{e_1 \rightarrow e_2 \equiv e_1' \rightarrow e_2'}{e_1 \equiv e_1' \ \wedge \ e_2 \equiv e_2'}$$

In egg♯, $\rightarrow$ is represented as a relation with three columns e_1, e_2, and c, and the above rule is equivalent a functional dependency from c to e_1 and e_2. Unification is known in the literature to be the dual to congruence [8]. The functional dependency captures this duality: reversing the arrows in functional dependencies in a congruence theory produces its unification dual.

Based on the experience with egg♯, we started a potential design for egg 2.0, which we call egg♯[2]. egg♯ is an extension to Datalog that provides a unified language for describing congruence reasoning and e-class analyses based on functional dependencies. Relations in egg♯ are annotated with functional dependencies. Values in egg♯ are divided into sorts and lattices. Sorts are uninterpreted and values of the same sort can be unified. Relations whose dependent sets (columns determined by other columns) have only sort values generalize e-classes in an e-graph, and relations whose dependent sets have only lattice values generalize e-class analyses. In other words, in egg♯, e-classes and e-class analyses are just relations with different dependent sets.

When two values of the same sort are unified, they are no longer distinguishable in egg♯. Such unification could potentially break the integrity of functional dependencies, i.e., multiple distinct tuples with the same determinant set (columns that determine other columns) can exist after unification. egg♯ remedies these violations. For each sort column in the dependent set, egg♯ unifies the sort values in that column, which makes values in the column indistinguishable and therefore unique again. For each lattice column in the dependent set, the new, unique value for each column is computed as the join of the lattice values in that column. As a result, functional dependency repair in egg♯ unifies the semantics of e-classes and e-class analyses.

egg♯ greatly expands the expressivity of egg: It has natural support of multi-patterns thanks to the relational representation. As an extension to Datalog, it also supports various kinds of reasoning expressible in Datalog, including non-symmetric ones. Finally, lattice values in egg♯ generalize e-class analyses. As an example, egg♯ allows interdependent analyses, which are naturally expressed as egg♯ rules over several analysis relations. In contrast, e-class analyses in egg are not composable. With the new expressive power, egg♯ is able to express the classical type inference algorithm for Hindley-Milner type systems.

egg♯ also benefits from the efficient evaluation algorithms in Datalog. For example, a straightforward semi-naïve evaluation algorithm for egg♯ exists, which may be unintuitive or inefficient in the traditional representation of e-graphs.

In the talk, I will first describe issues with the current egg interface. Next, I will introduce egg♯ and describe the algorithms that make it practical. In particular, I will go through the chase-like algorithms for match-apply iterations in equality saturation and the rebuilding algorithm in egg♯. I will also go through some alternative designs in egg♯. Finally, I will present egg♯'s early design, some example egg♯ programs, and its evaluation algorithms.

[2]Pronounced as egg sharp.

References

[1] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) *(PODS '17)*. Association for Computing Machinery, New York, NY, USA, 37–52. https://doi.org/10.1145/3034786.3034796

[2] Alessandro Cheli. 2021. Metatheory.jl: Fast and Elegant Algebraic Computation in Julia with Extensible Equality Saturation. *Journal of Open Source Software* 6, 59 (2021), 3078. https://doi.org/10.21105/joss.03078

[3] Neil Conway, William R. Marczak, Peter Alvaro, Joseph M. Hellerstein, and David Maier. 2012. Logic and lattices for distributed programming. In *ACM Symposium on Cloud Computing, SOCC '12, San Jose, CA, USA, October 14-17, 2012*. 1. https://doi.org/10.1145/2391229.2391230

[4] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. 1980. Variations on the Common Subexpression Problem. *J. ACM* 27, 4 (oct 1980), 758–771. https://doi.org/10.1145/322217.322228

[5] Kiran Gopinathan. 2021. Ego - E-graphs in OCaml. https://github.com/verse-lab/ego

[6] Todd J. Green, Shan Shan Huang, Boon Thau Loo, and Wenchao Zhou. 2013. Datalog and Recursive Query Processing. *Found. Trends Databases* 5, 2 (Nov. 2013), 105–195. https://doi.org/10.1561/1900000017

[7] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Beijing, China) *(PODS '07)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/1265530.1265535

[8] Paris C. Kanellakis and Peter Z. Revesz. 1989. On the Relationship of Congruence Closure and Unification. *J. Symb. Comput.* 7, 3/4 (1989), 427–444. https://doi.org/10.1016/S0747-7171(89)80018-5

[9] Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. 2021. Convergence of Datalog over (Pre-) Semirings. *CoRR* abs/2105.14435 (2021). arXiv:2105.14435 https://arxiv.org/abs/2105.14435

[10] Magnus Madsen, Ming-Ho Yee, and Ondřej Lhoták. 2016. From Datalog to Flix: A Declarative Language for Fixed Points on Lattices. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. Association for Computing Machinery, New York, NY, USA, 194–208. https://doi.org/10.1145/2908080.2908096

[11] Chandrakana Nandi, Max Willsey, Adam Anderson, James R. Wilcox, Eva Darulova, Dan Grossman, and Zachary Tatlock. 2020. Synthesizing Structured CAD Models with Equality Saturation and Inverse Transformations. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 31–44. https://doi.org/10.1145/3385412.3386012

[12] Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, and Zachary Tatlock. 2021. Rewrite Rule Inference Using Equality Saturation. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 119 (oct 2021), 28 pages. https://doi.org/10.1145/3485496

[13] Patrick Nappa, David Zhao, Pavle Subotić, and Bernhard Scholz. 2019. Fast Parallel Equivalence Relations in a Datalog Compiler. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 82–96.

[14] Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. 2015. Automatically Improving Accuracy for Floating Point Expressions. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI '15)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/2737924.2737959

[15] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. Egg: Fast and Extensible Equality Saturation. *Proc. ACM Program. Lang.* 5, POPL, Article 23 (Jan. 2021), 29 pages. https://doi.org/10.1145/3434304

[16] Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. 2021. Equality Saturation for Tensor Graph Superoptimization. *arXiv e-prints*, Article arXiv:2101.01332 (Jan. 2021), arXiv:2101.01332 pages. arXiv:2101.01332 [cs.AI]

[17] Yihong Zhang, Yisu Remy Wang, Max Willsey, and Zachary Tatlock. 2022. Relational E-Matching. *Proc. ACM Program. Lang.* 6, POPL, Article 35 (jan 2022), 22 pages. https://doi.org/10.1145/3498696