E-graphs and Automated Reasoning

Looking back to look forward

Anonymous Author(s)

Abstract

Automated reasoning [5] is an established field with a long history and many ideas. E-graph rewriting techniques fits into this history. This connection suggests roads to implementing fundamental extensions to e-graph rewriting.

Keywords: e-graphs, automated reasoning

1 Introduction

Term simplification is a natural and intuitive concept. It can be used to both solve algebra problems and optimize compiler output.

A first simple but effective impulse is to treat simplification via greedy rewriting.

Like many heuristic greedy methods, it is possible that a locally greedy move can lead to globally suboptimal results. It is sometimes necessary to travel up the mountain to escape the valley. To fix this, some amount of breadth first or backtracking exploration is necessary.

The field of automated reasoning has been aware of and refining these notions for a long time.

- Paramodulation [10] is a breadth first technique for (possibly conditional) equational reasoning
- Completion[2] is a method to convert an equational system to one with good guaranteed greedy rewriting properties. It can also be seen as equational reasoning with strong redundancy removal principles.
- Superposition [11] combines properties of completion and paramodulation.

E-graphs [15] are a compact data structure for representing a collection of ground equations over terms and their congruence closure. They are an intuitively appealing technique for theorem proving and simplification. No small part of this intuition is provided by the bipartite graphical diagram of e-classes and e-nodes. There is a lower theoretical and technical barrier to entry in e-graph rewriting than for navigating the complex automated reasoning literature. This conceptual simplicity has been essential for the e-graph's flourishing applications.

There is also a difference in attitude between the mainstream of the automated reasoning community and the e-graphs community. The majority of the literature in automated reasoning is concerned with theorem proving problems. The e-graph literature has a healthy contingent concerned with simplification/optimization/-compiler problems. The e-graph community wants a controllable quasi operational view of the search process and desire for completeness of the search does not override pragmatism. This is similar to the viewpoint of the Argonne school of automated reasoning [8] [7], which supported a more tweakable operational approach to automated reasoning.

This talk is about explaining some of the connections between different techniques and how they may give clues to fundamental problems in e-graphs.

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

2 Union Finds as Completion

The union find is a key component of the e-graph. It can be seen as an instance of completion [19]. Indeed many algorithms, including Gaussian elimination, Grobner bases, resolution and others, can be viewed in this light.

A union find is a forest of equivalence classes. The pointers in the tree point upward to parents rather than downward to children. This can be viewed as a discrete dynamical system describing a convergent flow to a canonical member of the class. In other words, a convergent rewriting system.

For example, the following rewrite system is a representation of a union find where b and c are children of a canonical member a and e is a child of the canonical member d.

$$b \to a$$
$$c \to a$$
$$e \to d$$

Completion can be described as an inference rule system [2], orienting equations into rewrite rules and simplifying them. This simplification is a perspective on the path compression step in the union find.

The pieces of a simple union find algorithm can be dignified via reference to these rules.

```
class UF():
  def __init__(self):
      self.rules = {}
  def find(self, x):
      # 'find' reduces x with respect
      # to the current rules (x -R-> retval)
      while self.rules.get(x) != None:
          x = self.rules.get(x)
      return x
  def union(self, x, y):
      # Do incremental completion starting with
```

```
        # (E,R) == ({x = y}, self.rules )
        x1 = self.find(x) # SIMPLIFY  ( {x1 = y} , R)
        y1 = self.find(y) # SIMPLIFY  ( {x1 = y1}, R)
        if x1 == y1: # TRIVIAL ({x1 = x1}, R)
            return x1 # (Empty, self.rules)
        else:
            if x1 < y1: # the "term order"
                x1,y1 = y1,x1 # swap
            # ORIENT  (empty, R U {x1 -> y1})
            self.rules[x1] = y1
            return y1
    def canon(self):
        for lhs,rhs in self.rules.items():
            self.rules[lhs] = self.find(rhs) # r-simplify
```

3 E-graphs as completion

Performing a regular completion procedure on ground terms is guaranteed to terminate. By putting the system in a flat canonical form, it is easier to see the correspondence with more familiar e-graph notions.

The flattening transformation can be achieved by creating fresh ground symbols for every function application in the original set of equations.

For example,

$$foo(biz(baz), bar) = bar$$

becomes

$$bar = e1$$
$$baz = e2$$
$$biz(e2) = e3$$
$$foo(e3, e1) = e4$$
$$e4 = e1$$

With an appropriate ground term ordering that puts every e less than any function symbol will orient the system into the form

$$bar \to e1$$
$$baz \to e2$$
$$biz(e2) \to e3$$
$$foo(e3, e1) \to e4$$
$$e4 \to e1$$

Completing a ground system of this sort will put it in a canonical form. The rewrite rules can be separated into two classes, those that rewrite e-nodes to e-classes and those that rewrite e-classes to other e-classes. The first represents the membership of e-nodes to their e-class and the second class represents a union find.

If we write e as q, this becomes the definition of a tree automata as has been noted [16]. Tree automata implicitly describe classes of trees using functional folds over finite state accumulators as acceptor functions, which can be described by tabulatable data. The ground rewriting point of view of e-graphs is the same observation as identifying e-graphs with tree automata but with a different emphasis of ideas.

The flattening transformation is not necessary for ground completion, but it does make the system more uniform. Notions of binding and context can be an impediment to this flattening transformation, so this complication may have a purpose.

3.1 Extraction

The idea of a term ordering has not made an explicit appearance in the modern e-graphs literature to the author's knowledge. Term ordering is a generalization of the notion of comparing terms for simplicity [2]. We typically want to rewrite a big term into a smaller one. With a completed ground rewrite system, running the system against the initial term will compute an equivalent smaller term. This is the analog of the e-graph extraction process, and picking a term order picks the extraction goals. Giving symbols extraction weights corresponds to a similar notion that appears in knuth bendix ordering.

4 E-matching

The first component of e-graph rewriting is to have an e-graph. The second component is e-matching.

An e-graph implicitly represents a possibly infinite set of terms that are equivalent to terms inserted via the ground equations.

A ground completed rewrite system represents a possibly infinite set of terms that rewrites to something that is a right hand side of the system. Alternatively, the set is generated by running the rules backwards.

E-matching is trying to find a term in this set that matches a pattern. This can be described in the terminology of the GRS without explicit reference to e-graphs [18].

Bottom up e-matching is the simplest naivest method. One makes a nondeterministic guess to fill in variables in the pattern with a choice of right hand side in the GRS. Then one reduces the grounded pattern and see if reduces to a right hand side or not. Top down e-matching can be achieved by narrowing the variables in the pattern according to current rewrite system. Every narrowing will ground out the pattern, since the rewrite system is ground.

5 Saturation

Saturation is not a unique property of e-graph rewriting. Traditional automated reasoners are also saturating systems.

It is a known technique in the automated reasoning community that it is useful to separate clauses into at least two groups that are not treated symmetrically.

Sometimes these are called unprocessed/processed or usable/set of support [8]. Naive resolution or paramodulation considers every possibly pair of interaction between currently derived clauses. The given-clause algorithm is a form of semi-naive evaluation that only further processes fresh clauses from the unprocessed set against the processed set.

Something quite similar to e-graph rewriting can be made to occur by restricting a superposition prover to only allow superposition between a ground rewrite rule and a non-ground rewrite rule. This is similar to some mixture of hyper-resolution, UR-resolution, and set of support strategies [7].

6 Hints for a Road Forward

There are a number of possible tantalizing expressivity extensions to the e-graph. The extreme sharing and destruction of context in the e-graph make many extensions hard to talk about. The connection back to term rewriting and automated reasoning sheds light on possible implementation methods.

6.1 Context

Superposition is a technique for equational theorem proving. It extends Knuth Bendix completion to equational clauses. A clause of terms $a \neq b \vee b = c$ can be seen as an implication $a = b \rightarrow b = c$.

Horn encodings [3] are a technique for getting context into pure completion solvers. It is similar in character to ASSUME nodes [4].

Another suggested technique for context is colored e-graphs [12]. I am not aware of a similar notion in the automated reasoning literature.

6.2 Lambda

Recently superposition has been extended to support lambda terms [1]. A restriction of this capability should be sufficient to implement an analog of e-graph rewriting for lambda terms.

It is not clear that full lambda unification or matching is necessary or desirable. It depends on the application and performance trade offs. More limited but efficient forms of matching that nevertheless treat variable binding correctly such as Nominal unification [13] or Miller patterm unification [9] may be preferred.

6.3 Backwards Reasoning

E-graph saturation is a bottom up technique analogous to datalog. A natural question is raised as to what a natural top down like prolog would look like.

Datalog and prolog can be viewed as incomplete strategies for resolution in a similar way that egglog [17] and functional logic programming are incomplete strategies for superposition.

Understanding this connection may be useful in possible applications of e-graph techniques to typeclass or trait resolution.

6.4 AC

The automated reasoning community has been aware of the issue of associativity and commutativity from the beginning. These rewrite rules are difficult to orient, commonplace, and very structural. There is a large literature on this subproblem and modern automated theorem such as E and Vampire have some intrinsic support for this.

6.5 Eager Rewriting

Demodulation is terminology used for eager rewriting in theorem provers. The ground rewrite system perspective of e-graphs lends some clarity to the lack of completeness that may result.

E-matching in the e-graph's community refers to matching over an e-graph. E-matching in the automated reasoning community refers to the more general mechanism of matching with regards to a background equational theory, often theories that are confluent and terminating. This is another approach to building in rewriting with respect to an a priori well behaved set of rewrite rules.

6.6 Universal Variables

Proving universal goals using e-graphs requires some slight of hand and preprocessing. Standard automated reasoning techniques support true unification variables, which are an assertion of the universal applicability of the fact derived. They are scoped to their clause and are alpha renameable, which makes them distinct from the e-graph's equational constant symbols. Some degree of inter-emulation is possible.

6.7 Sketches and Hints

The two communities have separately invented similar notions of sketches [6] and hints [14].

7 Conclusion

E-graph rewriting and other automated reasoning techniques are deeply interconnected. Neither is strictly better than the other and lessons can flow in both ways. The e-graph has a intuitive appeal and by focusing on a subproblem is simpler to implement and hence fast. Automated reasoning systems implementing completion or superposition have a long history of theoretical and implementation refinement. It is seemingly conceptually simpler to extend these systems than to extend the highly shared and context destroying e-graph, but perhaps a

new efficient and expressive perspective can emerge by pumping insight between the two.

References

[1] Sophie Tourret Petar Vukmirović Alexander Bentkamp, Jasmin Blanchette and Uwe Waldmann. 2021. Superposition with Lambdas. *Journal of Automated Reasoning* 65, 5 (2021), 893–940. https://doi.org/10.1007/s10817-021-09595-y Received: 15 April 2020, Accepted: 24 January 2021, Published: 21 August 2021, Issue Date: October 2021.

[2] Franz Baader and Tobias Nipkow. 1998. *Term rewriting and all that.* Cambridge University Press, USA.

[3] Koen Claessen and Nicholas Smallbone. 2018. Efficient Encodings of First-Order Horn Formulas in Equational Logic. In *Automated Reasoning*, Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.). Springer International Publishing, Cham, 388–404.

[4] Samuel Coward, George A. Constantinides, and Theo Drane. 2023. Automating Constraint-Aware Datapath Optimization using E-Graphs. arXiv:2303.01839 [cs.AR]

[5] Martin Davis. 2001. Chapter 1 - The Early History of Automated Deduction: Dedicated to the memory of Hao Wang. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). North-Holland, Amsterdam, 3–15. https://doi.org/10.1016/B978-044450813-3/50003-5

[6] Thomas Kohler, Andrés Goens, Siddharth Bhat, Tobias Grosser, Phil Trinder, and Michel Steuwer. 2024. Guided Equality Saturation. *Proc. ACM Program. Lang.* 8, POPL, Article 58 (jan 2024), 32 pages. https://doi.org/10.1145/3632900

[7] Rusty Lusk Larry Wos, Ross Overbeek and Jim Boyle. 1992. *Automated Reasoning: Introduction and Applications.* McGraw-Hill. https://books.google.com/books?id=H0g_AQAAIAAJ

[8] William McCune. [n.d.]. OTTER 3.3 Reference Manual. ([n. d.]). https://www.mcs.anl.gov/research/projects/AR/otter/otter33.pdf

[9] Dale Miller and Gopalan Nadathur. 2012. *Programming with Higher-Order Logic* (1st ed.). Cambridge University Press, USA.

[10] Robert Nieuwenhuis and Albert Rubio. 2001. Chapter 7 - Paramodulation-Based Theorem Proving. In *Handbook of Automated Reasoning*, Alan Robinson and Andrei Voronkov (Eds.). North-Holland, Amsterdam, 371–443. https://doi.org/10.1016/B978-044450813-3/50009-6

[11] Stephan Schulz. 2002. E - a brainiac theorem prover. *AI Commun.* 15, 2,3 (aug 2002), 111–126.

[12] Eytan Singher and Shachar Itzhaky. 2023. Colored E-Graph: Equality Reasoning with Conditions. arXiv:2305.19203 [cs.PL]

[13] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. 2004. Nominal unification. *Theoretical Computer Science* 323, 1 (2004), 473–497. https://doi.org/10.1016/j.tcs.2004.06.016

[14] Robert Veroff. 1996. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *Journal of Automated Reasoning* 16, 3 (1996), 223–239. https://doi.org/10.1007/BF00252178

[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] Yihong Zhang and Oliver Flatt. 2023. Ensuring the termination of equality saturation for terminating term rewriting systems. In *E-Graph Research, Applications, Practices, and Human-factors Symposium (EGRAPHS 2023)*. ACM, New York, NY, USA. https://pldi23.sigplan.org/details/egraphs-2023-papers/9/Ensuring-the-termination-of-equality-saturation-for-terminating-term-rewriting-system

[17] Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. *Proc. ACM Program. Lang.* 7, PLDI, Article 125 (jun 2023), 25 pages. https://doi.org/10.1145/3591239

[18] Philip Zucker. 2023. E-graphs are Ground Rewrite Systems 2: E-matching. https://www.philipzucker.com/ground-rewrite-2/

[19] Philip Zucker. 2023. A Road to Lambda: E-graphs are Ground Completion. https://www.philipzucker.com/egraph-ground-rewrite/