Hatching a Datalog:

Horn Clauses Modulo Equality using Egg

Anonymous Author(s)

Abstract

There are many striking similarities between the worlds of term rewriting, logic programming, and databases. It can be nontrivial but useful to translate insights between them. This talk is about a prototype implementation Egglog0 built upon the e-graph framework library egg and the modelling power equality saturation gains when extended to logic programming constructs.

Keywords: e-graphs, datalog, theorem proving

1 Syntax and Semantics of Egglog0

Egglog0 is a prototype system built around the egg equality saturation library. It extends egg with a naive implementation of multipatterns and is designed bring to the forefront the analogy between equality saturation and datalog.

Similar to datalog, ground terms can be asserted as facts.

```
edge(a,b).
edge(b,c).
add(succ(zero), succ(succ(zero))).
```

Operationally this inserts the ground term into the backing e-graph. To simplify the implementation, there is no distinction between the storage of predicates and terms. Hence, base facts can also be used to initialize the e-graph for equality saturation.

Clauses are represented using prolog convention of capitalized names for variables.

```
path(X, Y) :- edge(X, Y).
path(X, Y) :- path(X, Z), edge(Z, Y).
```

Operationally, a term that appears on the right is a multipattern to seek in the e-graph. A ground term is produced and inserted into the e-graph by instantiating the pattern on the left.

The features so far have not used the e-graph in any intrinsic way. This subset of the language, "hashlog", could be implemented by a backing hash cons data structure instead and is available in a datalog that supports algebraic datatypes.

Egglog0 also possesses a special built in relation _=_ backed by e-graph equality. Base facts, clause heads, and clause bodies may use _=_. A `p1 = p2` in the body corresponds to a check that the patterns `p1` and `p2` were found in the same class, In the head of a clause it correspond to called ‘union¡ on two eclasses of the instantiated terms.

Pattern variables are best not thought of as binding to terms, but instead to e-classes. Rewrite rules can be encoded as Horn clauses.

```
add(Y,X) = E :- add(X,Y) = E
```

Because unidirectional and bidirectional rewriting is a common use case, there is custom syntax sugar for it.

```
add(X,Y) <- add(Y,X).
add(X,add(Y,Z)) <-> add(add(X,Y),Z).
```

To recover information out of the e-graph, there are queries. Queries are run after (possibly early) termination of the equality saturation process. They are implemented as multipatterns followed by extraction. Internally, a substitution dictionary of variables to eclasses is returned from the multipattern. To print out a concrete ground term, a small representative term is extracted out of the variables eclass.

```
?- add(succ(zero),succ(Y)) = Z.
```

2 Applications

Why bother with this extra expressivity? In this section we demonstrate a number of applications that are not easily expressible as equality saturation or datalog on their own.

2.1 Injectivity

One simple application is being able to express the axiom of injectivity of a constructor. This axiom is in some sense the dual of the congruence axiom which is applied automatically in the e-graph. Here we also see the multiple heads feature of Egglog0, which performs both heads effects upon a successful multipattern match from the body.

```
X = Y, Xs = Ys :- cons(X,Xs) = cons(Y,Ys).
```

2.2 Equation Solving Rules

A common manipulation in algebraic reasoning is to manipulate equations by applying the same operation to both sides. This is often used for the purposes of variable isolation and then substitution to achieve variable elimination. The

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

e-graph by its very nature immediately understands the substitution, but we must show it how to manipulate equations. As an example, the following axiom isolates the Y variable.

```
sub(Z,X) = Y :- add(X,Y) = Z
```

It is possible to extract from the e-graph a term containing the least number of undesirable variables that are in the same class as a term `t`, although Egglog0 does not currently implement this extraction.

2.3 Reflection and Equational Logic

Like datalog, the presence of predicate "terms" in the e-graph database can be considered as an assertion of their truth. The alternative convention of modeling predicates as functions into `Bool` allows one to manipulate and talk about predicates in the absence of knowledge of their truth. Instead, a predicate term being in the same eclass as the term `true` can be seen as evidence of its truth. More interestingly, the lack of a predicate being in the e-graph cannot be seen as evidence of it's falsehood, whereas being in the same eclass as `false` can be seen as evidence that the predicate has been determined to be definitely false. Equality saturation as a reasoning tool has constructive character in this sense. This is the analog of the situation in datalog, where negation may only be considered in the presence of a stratification condition. This encoding allows for ordinary boolean algebraic manipulation of logical formula. One can reflect the notion of e-graph equality itself into a term `eq(_,_)`.

```
A = B :- true = eq(A,B).
true = eq(A,B) :- A = B.
```

The second rule is possibly very expensive, with an unconstrained search and creating many new terms. The first rule will however typically compress the e-graph.

For further material on equality reasoning over equality itself see [reference to dijkstra book, de gries equational logic]

2.4 Uniqueness Quantification

Equality generating dependencies and tuple generating dependencies are two constraints from the theory of relational databases. Equality generating dependencies can be directly modeled using native e-graph =. Tuple generating dependencies can be modelled using Skolemization. An axiom of the form

$$\forall x_1, \ldots, x_n, \phi(x_1, \ldots, x_n) \implies \exists y_1, \ldots, y_m, \psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

can be converted into an axiom where $y_1 \ldots y_m$ are n-arity globally fresh function symbols.

```
psi(X1,..., Xn, y1(X1,...,Xn), ... , ym(X1, ...,
    Xn)) :- phi(X1, ... ,Xn)
```

Combining these two abilities allows one to express uniqueness quantification $\exists!$, which one finds for example in the definition of universal properties in category theory.

$$\forall x_1, ..., x_2.\psi(x_1, ..., x_n) \implies \exists! y1, \cdots, y4\psi(y_1, ...y_n)$$

$$\forall x_1, ..., x_2.\psi(x_1, ..., x_n) \implies \exists y1, .., ym\psi(y_1, ...y_m) \wedge \forall x_1, ..., x_n, y1, .., ym, x_1'$$

Diagrams chasing is considered to be difficult and yet fairly automatic activity. A large part of it consists of generating commuting squares (equalities), and finding places to instantiate universal properties. This search and instantiation can be encoded in Egglog0 clauses.

[reference to catnapp]

2.5 Generalized Algebraic Theories

[catlab and whatsies. the GAT guy cartmell]

Algebraic theories are those described by equational axioms. An example is the theory of groups. Generalized algebraic theories extend these by introducing a limited version of dependent types. The types also have an equational theory and the equational axioms only hold between terms of the same type, therefore there may be typing obligations to discharge before one can use a rewrite rule.

As an example, a monoidal category has morphisms of type Hom(a,b),

The typing requirements of a GAT axiom naturally map to horn clauses. It is possible to build an encoding using type tagging [type tagging paper] and guards rather than multipatterns, but it is extremely burdensome and inefficient.

TODO: possibly talk about kronecker product rewriting

3 Related Work

This work draws on the ideas of Relational E-matching Egglite is an implementation of similar functionality relying on an e-graph built on top of sql-lite Souffle is a datalog engine supporting algebraic datatypes, equivalence relations, and provenance production. These features make it possible to emulate Egglog0.

4 Further Work

The most tantalizing of all are the places where we don't find obvious analogs of techniques of one domain in the other. - semi naive - Proofs - Efficient implementation - Magic Set

Harrop Formula How to internalize Analysis, Extraction

A Appendix

Text of appendix ...