Hatching a Datalog:

Horn Clauses Modulo Congruence (Equality?) using Egg

Anonymous Author(s)

Abstract

Text of abstract ....

Keywords: keyword1, keyword2, keyword3

1 Introduction

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.

2 Term Rewriting and Logic Programming

I don't have space for this.

3 Egglog0

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

Similar to datalog, base facts are ground terms.

```
human(socrates).
```

Operationally this corresponds to inserting this ground term into the backing e-graph. To simplify the impementation, there is no distinction between predicates and terms. This is odd but can also be useful.

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

TODO: use example that has terms in it, not just datalog

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

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

NOTE: show multiheads?

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

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 possess special built in relation = backed by e-graph equality. Base facts may be inserted into the e-graph.

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

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

Operationally, an = the body corresponds to a check that the patterns were found in the same class, in the head of a clause correspond to called ‘union¡ on two eclasses of the constructed terms. In principle = could be used in a ‘-+‘ mode, efficiently generating members of an eclass, however the details of the egg interface prevent this as a user of the library.

Because standard equality saturation is a common use case, there is custom syntax sugar for it and also bidirectional rewriting, which expands to two unidirectional rewriting clauses.

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

Finally, 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 via the exact same mechanism as multipatterns. Internally, a substitution dictionary of variables to eclasses is returned. To print out a concrete ground term, a small representative term is extracted out of the variables eclass.

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

4 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.

4.1 Injectivity, unification, and Eta Rules

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.

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

A related axiom is that describing the properties of projectors of a pair.

```
pair(A,B) = Z :- proj1(Z) = A, proj2(Z) = B.
proj1(Z) = A, proj2(Z) = B :- pair(A,B) = Z.
```

From these rules, one can derive the computation rule

```
Z <- pair(proj1(Z), proj2(Z)).
```

However this rule is weaker and the left to right direction will never saturate and choke the e-graph with terms.

4.2 Equation Solving Rules

A common manipulation in algebraic reasoning is to manipulate equations by applying the same operation to both sides. This is not a rewrite rule persay, but instead manipulating the equation itself. This is often used for the purposes of variable isolation and then substitution. The e-graph by it's very nature immediately understands the substitution, but we must teach it the available manipulations of equations. As an example, the following axiom manipulates the Y variable to isolation.

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

Although Egglog0 does not currently implement this, it is in possible to extract from the egraph a term containing the least number of undesirable variables that are in the same class as a term y.

4.3 Reflection and equational logic

While like datalog, the presence of predicate "terms" in the e-graph database can be considered as an assertion of their truth, the convention of modeling predicates as functions into `Bool` allows one to manipulate and talk about predicates in the absence of knowledge of their truth. The existence of a predicate term in the same eclass as `true` is then seen as evidence of it's truth. More interestingly, the existence of being in the same eclass as `false` can be seen as evidence that the predicate has determined not to be true. The analog of the lack of a predicate in the e-graph cannot be seen as evidence of it's falsehood. Equality saturation as a reasoning tool has constructive character in this sense. This is the analog of the situation in datalog implementations, where negation may only be considered in the presence of a stratification.

This encoding allows for ordinary boolean algerbaic 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 egraph.

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

4.4 Uniqueness quantification

Equality generating dependencies and tuple generating dependencies are two constraints from the theory of database schema. They are a method by which to model functional dependencies and foreign keys among other things.

Equality generating dependencies can be directly modeled using native egraph =. Tuple generating dependencies can be modelled using Skolemization. An axiom of the form

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

Can be converted into an axiom where $y_1..y_m$ are n arity globally fresh function symbols (alpha renaming if necessary). $\forall x_1, \ldots, x_n.\phi(x_1, ..., x_n) \longrightarrow \psi(x_1, \ldots, x_n, y_1(x_1, \ldots, x_n), ...y_m(x_1, \ldots, x_n))$
Which in turn is modellable in Egglog0 as

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

As pattern variables are implicitly universally quantified.

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

∃!.

$$\forall x_1, ..., x_2.\psi(x_1, ..., x_n) \longrightarrow \exists! y1, y2, y3, y4 \psi(y_1, ... y_n)$$

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

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]

4.5 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

5 Related Work

Egg-lite Souffle ATP SMT

6 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 …