15-745 Project Progress Report: Implementing Dataflow Optimizations for Pegasus in Datalog

Dan Licata

Carnegie Mellon University

Dataflow Analyses, Traditionally

- Lots of C++ code
- Same code over and over (e.g., iterate to fixed point)
- Hard to reason about directly
- Have to hand-tune for performance

Project Thesis

Using logic programming, you can easily easily implement correct and fast dataflow analyses for the Pegasus IR.

Logic Programming in Datalog

Finite data types:

```
NODE 1024
```

and relations on them:

```
edge  (x:NODE, y:NODE)
reach (x:NODE, y:NODE)
```

Logic Programming in Datalog

Some relations are specified explicitly:

```
edge(0,1).
edge(2,3).
...
```

Others are defined by inference rules:

```
reach(x,y) :- edge(x,y).
reach(x,z) :- edge(x,y), reach(y,z).
```

Computation: saturate database with all true facts.

Reasoning about Logic Programs

```
reach(x,y) :- edge(x,y).
reach(x,z) :- edge(x,y), reach(y,z).
```

- Program has a logical meaning
- Can be used to prove correctness properties. Example: if `reach(x,y)` then there exists a path from x to y.

Fast Logic Programming in bddbddb

Whaley and Lam (Stanford) implemented Datalog using Binary Decision Diagrams:

- Use BDDs to concisely represent relations
- Applied to pointer analyses, etc.
- One fast BDD implementation is shared by all analyses.

Example: Liveness

Iterative database saturation models iterative dataflow solving:

```
live(r, s)  :- uses (s, r).
live(r, s1) :- succ(s1,s2),
               live(r,s2), !assigns(s2, r).
```

where

```
uses(s:Stat, r:Reg)    % s is r' := op(r, _) ..
assigns(s:Stat, r:Reg) % s is r := ...
succ(s1:Stat, s2:Stat) % s2 after s1
```

So Far

- Got going with bddbddb
- Implemented some simple analyses for traditional CFG IR: reachability (simplified ADCE), liveness, first parts of PRE
- Prelim results: reachability on 100 nodes in 1s, 1000 nodes in 30s-2min
- Technique for intersection-analyses (more on this if there's time)
- Working on Pegasus analyses

Union Analyses

Inference rules implicitly existentially quantify over variables in premises:

```
live(r, s1) :- [exists s2.]
               succ(s1,s2),
               live(r,s2), !assigns(s2, r).
```

Works for dataflow analyses that union over adj. nodes.

Intersection Analyses

E.g. globally anticipatable expressions (first part of PRE):

```
locAnt(b:Blk, e:Exp)
locTrans(b:Blk, e:Exp)
succ(b1:Blk, b2:Blk)

ant(b,e)  :- locAnt(b,e).
ant(b1,e) :- locTrans(b1,e),
             (all b2. succ(b1,b2)
                      -> ant(b2,e)).
```

Problem: Datalog does not allow `all, ->` in a premise.

Double-negation Encoding

Define the negation of the intersection analysis:

```
notant(b,e)  :- !locAnt(b,e), !locTrans(b,e).
notant(b1,e) :- !locAnt(b1,e),
                succ(b1,b2), notant(b2,e).
```

One more negation at the outside:

```
ant(b,e) :- !notant(b,e).
```

Works when the analysis uses intersections or unions but not both—to guarantee saturation, a relation cannot be defined in terms of its own negation.