OPTIMIZATION OF RECURSIVE QUERIES

Tomasz Pieciukiewicz
AGENDA

- Optimization of transitive closures
  - Factoring out independent subqueries
  - Pushing out selections
- Optimization of fixpoint systems
  - Stratified evaluation
  - Factoring out independent subqueries
  - Pushing out selections
  - Detection of non-recursive equations
  - Combined techniques
OPTIMIZATION OF TRANSITIVE CLOSURES
Factoring out independent subqueries

Possible to use with any non-algebraic operator, transitive closure is not an exception
 FACTORING OUT - EXAMPLE

- (Activity where name = "Planning")
  close by
  
  (outgoing.Transition.leadsTo.Activity where
   maxTime < \text{avg(Activity.maxTime)})

- \text{avg(Activity.maxTime) as } x\text{.}((Activity where
  name = "Planning")
  close by
  
  (outgoing.Transition.leadsTo.Activity where
   maxTime < x))
OPTIMIZATION OF TRANSITIVE CLOSURES

- Pushing out selections in front of an operator
  - Possible to use with **distributive** non-algebraic operators
  - close unique by and leaves unique by are NOT distributive
  - close by and leaves by are distributive
PUSHING OUT - EXAMPLE

- `Car.Components.Part` close by `Component.leadsTo.Part` where `model = "Testarossa"`

- `(Car where model = "Testarossa") . Components.Part` close by `Component.leadsTo.Part`
OPTIMIZATION OF FIXPOINT SYSTEMS
OPTIMIZATION OF FIXPOINT SYSTEMS

- Stratified evaluation
  - Semi-naive evaluation technique
  - Divide fixpoint system into strata, evaluate them in sequence
  - Assignment of equations to strata based on referenced variable names
Identification of strata using Kosaraju algorithm for identification of Strongly Connected Components of a graph

- Vertices represent equations
- Edges represent variable references
  - If equation e1 uses a variable defined by equation e2, graph contains edge from vertice e1 to vertice e2
fixpoint {  
  engine : struct{Part,integer} := (Part where name="engine") as x, 1 as howMany;  
  engineParts : struct{Part,integer}[1..*] := engine union  
    (engineParts.Component.((leadsTo.Part) as x,  
      (amount*howMany) as howMany));  
  final s : struct{Part,integer}[1..*] :=  
    ((distinct(engineParts.x) as y).  
      (y,sum(engineParts where x=y).howMany)));  
}
**OPTIMIZATION OF FIXPOINT SYSTEMS**

```plaintext
engine : struct{Part,integer} := (Part where name="engine") as x, 1 as howMany;

engineParts : struct{Part,integer}[1..*] := engine union

(engineParts.Component.((leadsTo.Part) as x,

(amount*howMany) as howMany));

final s : struct{Part,integer}[1..*] := ((distinct(engineParts.x) as y).

(y,sum(engineParts where x=y).howMany));
```
Factoring out independent subqueries

We may optimize individual equations as usual

We may also treat the entire equation system as a non-algebraic operator! (see sbql.pl)

As a consequence, we may try to factor out an independent subquery in front of the fixpoint system
DATALOG MAGIC SETS

- identify selections within recursion, which may be evaluated only once ("magic predicates")
- assign names to those selections, replace selections within recursion with call to the newly assigned names
- evaluate those selections before going into recursion
- Does it look familiar?
OPTIMIZATION OF FIXPOINT SYSTEMS

- **fixpoint** {
  - **engine** : struct{Part,integer} =(Part where name="engine") as x, 1 as howMany;
  - **engineParts** : struct{Part,integer}[1..*] = engine union (engineParts.Component.((leadsTo.Part) as x, (amount*howMany) as howMany) where howMany>avg(Part.amount);
  - **final s** : struct{Part,integer}[1..*] = ((distinct(engineParts.x) as y). (y,sum(engineParts where x=y).howMany));
- }
OPTIMIZATION OF FIXPOINT SYSTEMS

(\text{avg}(\text{Part}.\text{amount}) \text{ as } t).\text{fixpoint} \{ \\
\text{engine} : \text{struct}\{\text{Part},\text{integer}\} := (\text{Part where name} = \text{"engine"}) \text{ as } x, 1 \text{ as } \text{howMany}; \\
\text{engineParts} : \text{struct}\{\text{Part},\text{integer}\}[1..*] := \text{engine union (engineParts}.\text{Component}.((\text{leadsTo}.\text{Part}) \text{ as } x, (\text{amount} \ast \text{howMany}) \text{ as } \text{howMany}) \text{ where howMany} > t; \\
\text{final s} : \text{struct}\{\text{Part},\text{integer}\}[1..*] := ((\text{distinct(engineParts}.x) \text{ as } y). (y,\text{sum}((\text{engineParts where } x=y).\text{howMany}))); \\
\}

OPTIMIZATION OF FIXPOINT SYSTEMS

- Pushing out selections
  - Only optimization of individual equations is possible
  - The semantic equivalence between fixpoint system and (non-distributive) leaves unique by operator by makes the use of this technique impossible
OPTIMIZATION OF FIXPOINT SYSTEMS

- Non-recursive equations within fixpoint systems
  + Some equations will return the same result in every evaluation
  + Usually they'll be evaluated twice – but why bother?
    ❔ Is it possible to easily identify them and evaluate them only once?
OPTIMIZATION OF FIXPOINT SYSTEMS

- If an equation references its own results from previous iteration – by using its name – it will of course have to be evaluated recursively.
- If an equation references the results of another equation, we have two possibilities:
  - the equation references another equation from the same stratum – in this case it will have to be evaluated recursively, as the results of the referenced equation may change in subsequent iterations.
  - the equation references only equations from previously evaluated strata – in this case the information required to evaluate the equation is available in the first evaluation, no recursive evaluation is necessary.
  - by definition, it is impossible for an equation to reference equation variable calculated in a stratum further in the evaluation order.
OPTIMIZATION OF FIXPOINT SYSTEMS

- This means, that an equation needs to be evaluated only once if it satisfies the following conditions:
  - it does not reference itself
  - it does not reference other equations in the same stratum
OPTIMIZATION OF FIXPOINT SYSTEMS

fixpoint {
  engine : struct{Part,integer} := (Part where name="engine") as x, 1 as howMany;
  allParts : struct{Part,integer}[1..*] := engine union
    (allParts.Component.((leadsTo.Part) as x, (amount*howMany) as howMany) ;
  engineParts : struct{Part,integer}[1..*] := engine union
    (engineParts.Component.((leadsTo.Part) as x, (amount*howMany) as howMany) where
      howMany<avg(allParts.howMany);
  final s : struct{Part,integer}[1..*] := ((distinct(engineParts.x) as y).
    (y,sum(engineParts where x=y).howMany));
}
OPTIMIZATION OF FIXPOINT SYSTEMS

- Combined techniques
  - A subquery that's not independent of the fixpoint system may be independent in the context of a particular stratum
  - We can't factor it out in front of the equation
    - But we can factor it out in front of the stratum!
fixpoint {
    engine : struct{Part,integer} :=(Part where name="engine") as x, 1 as howMany;
    allParts : struct{Part,integer}[1..*] := engine union
        (allParts.Component.((leadsTo.Part) as x,
            (amount*howMany) as howMany) ) ;
    engineParts : struct{Part,integer}[1..*] := engine union
        (engineParts.Component.((leadsTo.Part) as x,
            (amount*howMany) as howMany) where
            howMany<\text{avg}(allParts.howMany)};
    final s : struct{Part,integer}[1..*] := ((distinct(engineParts.x) as y).
            (y,sum(engineParts where x=y).howMany)));
}
fixpoint 

engine : struct{Part,integer} := (Part where name="engine") as x, 1 as howMany;

allParts : struct{Part,integer}[1..*] := engine union
  (allParts.Component.((leadsTo.Part) as x, (amount*howMany) as howMany) as howMany);

avg(allParts.howMany) as t;

distinct(engineParts) as y;

final s : struct{Part,integer}[1..*] := ((distinct(engineParts.x) as y). (y,sum(engineParts where x=y).howMany));
CONCLUSIONS

- Query rewriting techniques may be used for recursive queries
  - Although the use of pushing out selections is limited
- Stratification itself is useful
  - And it leads to further optimization possibilities
- There's no magic in "Magic sets"
  - Beyond the math used to describe them...