The AutoMed Query Processor

Previous Architecture

- Query reformulator: reformulates the input query *q* by following the transformation pathways from the GS to LS_i
- Fragment processor: replaces each scheme *s* with a wrapper containing *s*
- Evaluator: evaluates *q*

New Architecture

- Query reformulator: same functionality
- Logical optimiser: performs various logical optimisations
- Query annotator: detects the largest subtrees t_i translatable by the datasource wrappers and inserts a wrapper object as the root of t_i
- Physical Optimiser: performs datasource specific optimisations
- Evaluator: same functionality

Logical Optimisations

- Rule application using templates
  - Disjunction optimiser – see DBIS’04
  - Nil optimiser – see DBIS’04
  - Comprehensions optimiser. Example:

    [{h}|q_1; {x} **ß** (DS_1++DS_2); q_2] **à**

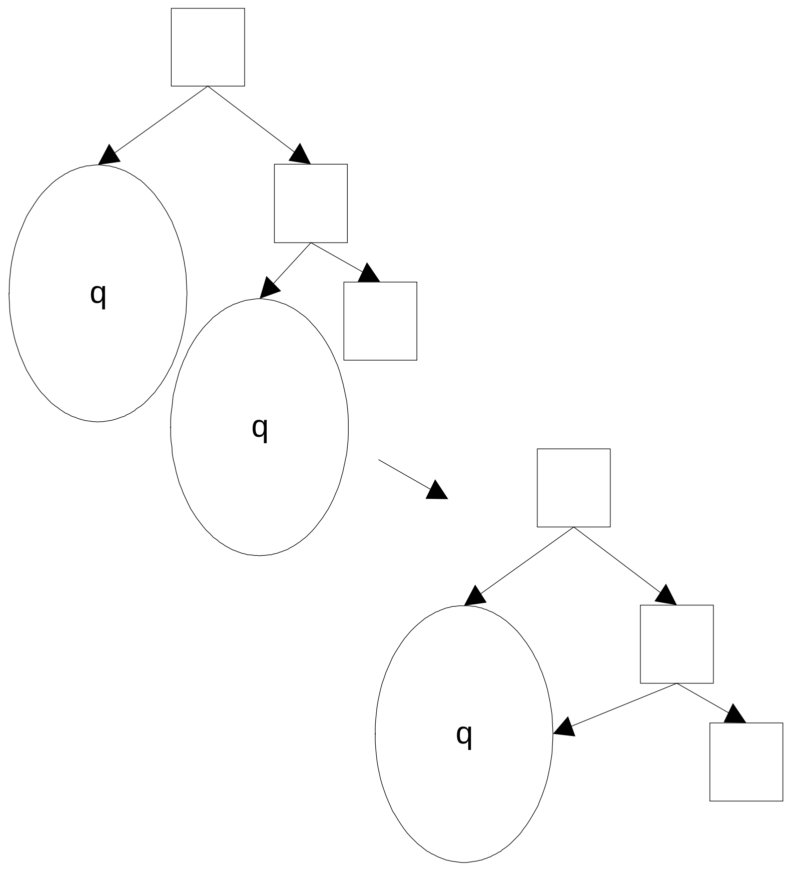
    [{h}|q_1; {x} **ß** DS_1; q_2] ++ [{h}|q_1; {x} **ß** DS_2; q_2]
- Java code to modify ASG structure
  - Datasource reorganiser
  - Common sub-expression elimination

Datasource Reorganiser

- Java-based optimisation
- Examples:
  - DS_1:A ++ DS_2:B ++ DS_1:C **à** (DS_1:A ++ DS_1:C) ++ DS_2:B
  - [{h}|DS_1:A;DS_2:C;DS_1:B;DS_2:D;p_1;p_2;p_{12}]
    à
    [{h}|{h_1} **ß** [{h_1}|DS_1:A;DS_1:B;p_1];
    {h_2} **ß** [{h_2}| DS_2:C; DS_2:D;p_2];
    p_{12}]

Common Sub-Expression Elimination

- Input query may contain multiple identical subqueries
- Transform input query into a DAG to avoid evaluation of the same subquery



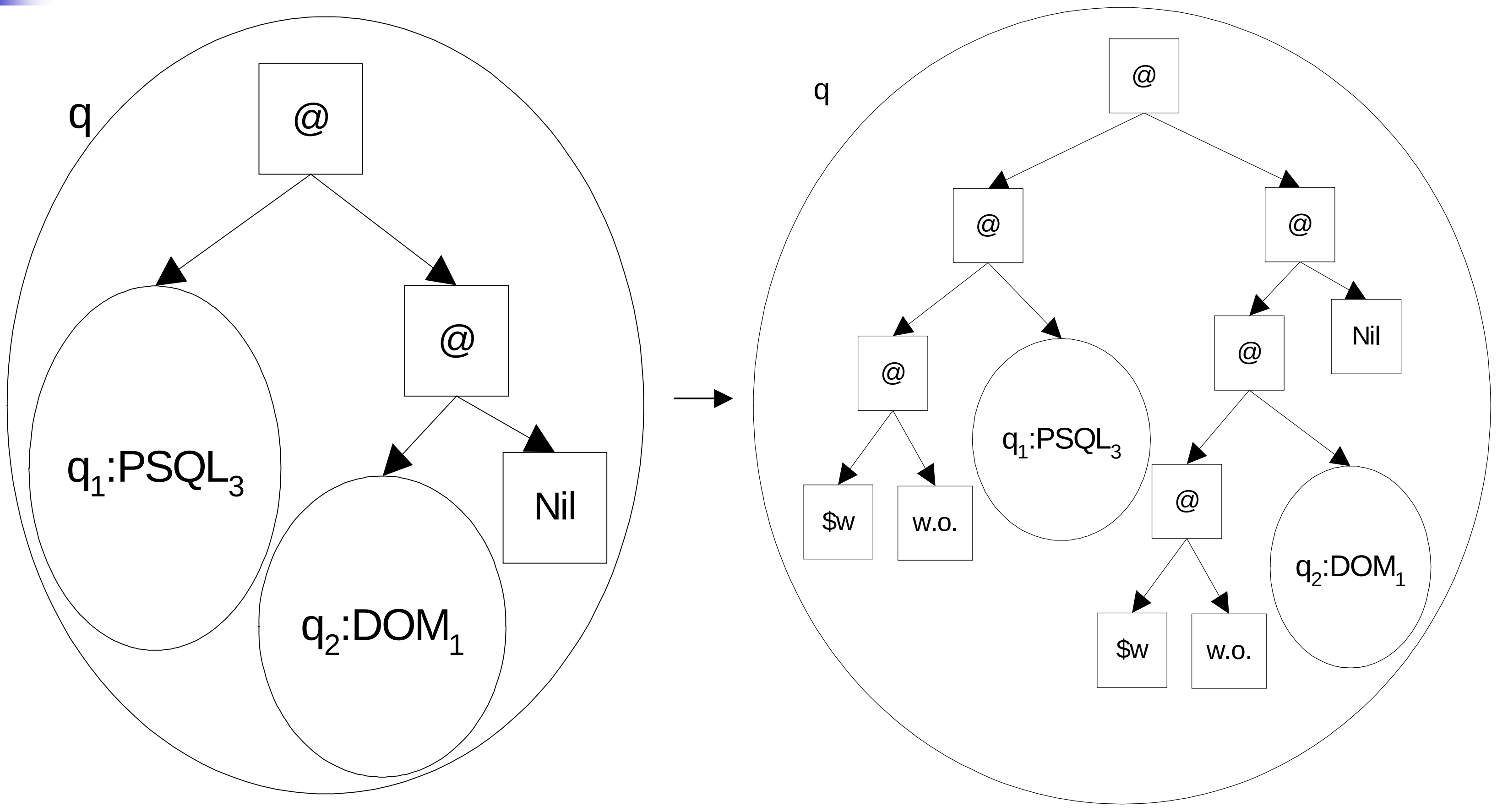
Logical Optimiser

- Applies logical optimisations using the following policy:
  - Step 1: apply each logical optimisation until an application does not modify the query
  - Step 2: apply step 1 until the query is not modified by any logical optimisation
  - Step 3: apply common sub-expression elimination

Query Annotator

- Detects the largest subqueries which can be translated by the datasource wrappers
- Once the annotator detects a translatable subtree, it inserts a wrapper object

Query Annotator - Example



Query Annotator – Detection

- Each wrapper is capable of translating a subset of IQL
- Each subset of IQL is represented in the query processor by a parser *p*
- When a query *q* is submitted to *p*, if it is not part of this subset of IQL, a syntax error is thrown
- Each wrapper defines the subset of IQL it can translate by selecting a parser; if no parser is selected, the default parser is used

Query Annotator – Detection

- Each Cell in the input query defines a subtree *t*
- The Query Annotator traverses the input query once for every datasource wrapper *w* and for every *t* checks whether it is translatable by *w*

Physical Optimiser

- Currently consists of a single optimiser, the dual model optimiser:
  - Some datasources are modeled using two modeling layers: datasource-oriented & AutoMed-oriented **à** may cause unnecessary self-joins of schema constructs:

<u>Original:</u>

```
    [{id,name}|{i,id} ß <<person,id>>;{i,name} ß <<person,dname>>]
```

<u>Reformulated:</u>

```
[{id,name}|{i,id} ß [{k1,k1}|{k1,a1,a2} ß <<person,3>>];
         {i,name} ß [{k1,a2}|{k1,a1,a2} ß <<person,3>>]]
```

<u>Optimised:</u> [{id,name}|{k,id,name} ß <<person,3>>]

External Functions

- Currently, the Evaluator handles only built-in and user-defined functions – all written in Java
- We are currently extending the evaluator to use external functions written in other programming languages, such as C, C++ and Perl

Lazy Evaluation

- The Evaluator currently fully evaluates a query submitted to it.
- This may be inadequate for queries that return large result sets which do not fit into the available memory.
- Thus, once the above functionality is fully implemented and tested, we will modify the evaluator to incrementally evaluate queries and to return fragments of result sets.

Type System – Type Checker

- Devise a unified type system for AutoMed
- Implement type checker which type checks input queries and queries supplied with transformations

Open Issues

- Dual model optimiser
- External functions
- Lazy evaluation
- Type system – type checker