A Uniform Calculus for Collection Types

Leonidas Fegaras
Oregon Graduate Institute

Problems with ODMG-93 OQL

- Implicit type conversions:

$$\begin{array}{l}\textbf{select } e \\ \quad \textbf{from } x_1 \textbf{ in } e_1,\ \ldots,\ x_n \textbf{ in } e_n \\ \quad \textbf{where } pred\end{array}$$

  OQL converts each e_i into a set before the join!
  flatten(x) is even messier.

- Missing operations:

$$\begin{array}{l}\textbf{select } \underline{\textbf{list}}\ e.B \\ \quad \textbf{from } e \textbf{ in } (\textbf{sort } s \textbf{ in } x \textbf{ by } s.A)\end{array}$$

- Redundancy:

$$flatten(x) \ = \ \textbf{select } a \textbf{ from } s \textbf{ in } x,\ a \textbf{ in } s$$

Fundamental questions

- Is the join of a list with a set meaningful?
- If so, what is the result type of this join?
- More generally, what are the precise semantics of queries over multiple collection types?
- How do we optimize such queries?

Extending the (nested) relational algebra is not necessarily a good solution

- some operations may have many instances and interpretations:
  e.g. join a list with a bag and return a set,
  e.g. flatten(x) returns a set if x is a set of sets,
  a bag if x is a bag of sets, etc;
  (EREQ/AQUA has about 100 operators!)
- the more bulk operations you support the harder the optimization task becomes;
  (need $\mathcal{O}(n^2)$ rules for n operations)
- testing program equivalences is more feasible for a small number of primitives;
- adding a new collection type will affect many operators.

Claims

- a query algebra doesn't have to resemble the query language or the physical algorithms;
- a query algebra should consists of a small number of powerful, generic, non-overlapping, operators (just one if possible).

But:
How these powerful operators are mapped into physical algorithms?

Motivation: Example from OQL

Cities : $set(\langle \text{name} : string, \text{hotels} : bag(\ldots), \text{places\_to\_visit} : list(\ldots)\rangle)$

OQL query:

select distinct h.name
from c **in** Cities,
h **in** c.hotels,
p **in** c.places_to_visit,
where c.name = "Madison" **and** h.name = p.name

In NF^2-style algebraic form:

$$\begin{aligned}&\textbf{project}(\textbf{flatten}(\textbf{map}(\textbf{filter}(\text{Cities}, \lambda c.\, c.\text{name} = \text{“Madison”}),\\ &\qquad\qquad \lambda c.\, \textbf{join}(c.\text{hotels},\ c.\text{places\_to\_visit},\\ &\qquad\qquad\qquad \lambda(h,p).\, h.\text{name} = p.\text{name})),\\ &\qquad \lambda(h,p).\, h.\text{name})\end{aligned}$$

In my calculus (comprehension syntax):

$$\begin{aligned}&set\{\, h.\text{name} \mid c \leftarrow \text{Cities},\ h \leftarrow c.\text{hotels},\ p \leftarrow c.\text{places\_to\_visit},\\ &\qquad\qquad c.\text{name} = \text{“Madison”},\ h.\text{name} = p.\text{name}\,\}\end{aligned}$$

Another example:

$$
\begin{array}{l}
\textbf{select } h.\text{name} \\
\quad \textbf{from } h \textbf{ in } (\textbf{select } c.\text{hotels} \\
\qquad\qquad\qquad \textbf{from } c \textbf{ in } \text{Cities} \\
\qquad\qquad\qquad \textbf{where } c.\text{name} = \text{“Madison”}) \\
\quad \textbf{where exists } r \textbf{ in } h.rooms : (r.bed\# = 3)
\end{array}
$$

$$
\begin{array}{l}
bag\{\, h.\text{name} \mid h \leftarrow bag\{\, c.\text{hotels} \mid c \leftarrow \text{Cities},\ c.\text{name} = \text{“Madison”} \,\}, \\
\qquad\qquad some\{\, r.bed\# = 3 \mid r \leftarrow h.rooms \,\}\,\}
\end{array}
$$

Monoids

$$(T, \text{zero}[T], \text{merge}[T])$$

is a **monoid** if merge$[T]$ is **associative** with a zero element:

$$\begin{array}{lcll} \text{merge}[T](\text{merge}[T](x,y),z) & = & \text{merge}[T](x,\text{merge}[T](y,z)) & \\ \text{merge}[T](\text{zero}[T],x) & = & \text{merge}[T](x,\text{zero}[T]) & = \; x \end{array}$$

Optional properties:

C) **Commutativity**: $\text{merge}[T](x,y) = \text{merge}[T](y,x)$

I) **Idempotence**: $\text{merge}[T](x,x) = x$

My approach: Every monoid is **predefined**!

free monoid T	zero$[T]$	unit$[T](a)$	merge$[T]$	C/I
list	$[\,]$	$[a]$	append	
set	$\{\}$	$\{a\}$	$\cup$	CI
bag	$\{\!\{\}\!\}$	$\{\!\{a\}\!\}$	$\uplus$	C
ordered-set	$\{[\,]\}$	$\{[a]\}$	$\dot{\cup}$	I

simple monoid T	type	zero$[T]$	unit$[T](a)$	merge$[T]$	C/I
sum	int	0	a	$+$	C
prod	int	1	a	$*$	C
max	int	0	a	max	CI
some	boolean	false	a	$\vee$	CI
all	boolean	true	a	$\wedge$	CI

Monoid Type:

$$
\begin{array}{lcll}
type & = & T & (T \text{ is a simple monoid}) \\
type & = & T(type') & (T \text{ is a free monoid}) \\
type & = & \langle a_1 : type_1, \ldots, a_n : type_n \rangle & (\text{aggregation})
\end{array}
$$

e.g.

Cities : $set(\langle \text{name} : string, \text{hotels} : bag(\ldots), \text{places\_to\_visit} : list(\ldots) \rangle)$

A monoid type is a monoid.

e.g.

$$
\begin{array}{lcl}
\text{zero}[T(type)] & = & \text{zero}[T] \\
\text{unit}[T(type)](x) & = & \text{unit}[T](\text{unit}[type](x)) \\
\text{merge}[T(type)](x, y) & = & \text{merge}[T](x, y)
\end{array}
$$

Monoid Homomorphisms

$\hom[T,S](f)$ is a **homomorphism** from monoid T to monoid S:

$$\begin{array}{lcl}
\hom[T,S](f)\,(\text{zero}[T]) & = & \text{zero}[S] \\
\hom[T,S](f)\,(\text{unit}[T](a)) & = & f(a) \\
\hom[T,S](f)\,(\text{merge}[T](x,y)) & = & \left\{ \begin{array}{r} \text{merge}[S](\,\hom[T,S](f)\,(x), \\ \hom[T,S](f)\,(y)\,) \end{array} \right.
\end{array}$$

If $T = list$ and $S = set$ then:

$$\begin{array}{lcl}
\hom[list,set](f)\,([\,]) & = & \{\} \\
\hom[list,set](f)\,([a]) & = & f(a) \\
\hom[list,set](f)\,(\text{append}(x,y)) & & \\
\qquad = \hom[list,set](f)\,(x) \cup \hom[list,set](f)\,(y) & &
\end{array}$$

e.g. if $x = [a_0, a_1, \ldots, a_n]$ then:

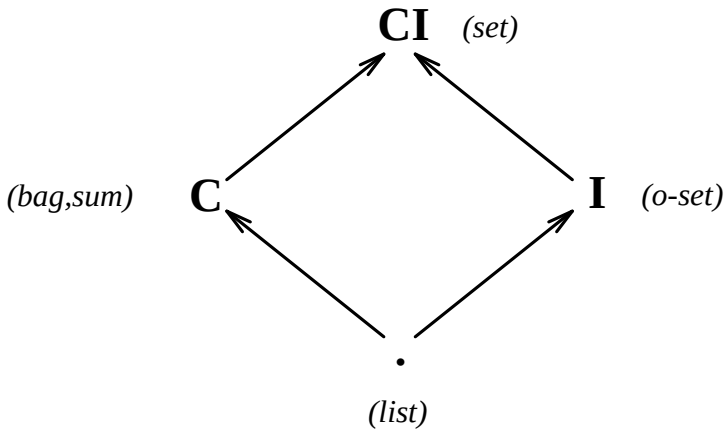
$$\hom[list,set](f)\,(x) = f(a_0) \cup f(a_1) \cup \cdots \cup f(a_n)$$

If $S = T$ then $\hom[T,T](f)$ is flattenmap(f) in [Wadler'90],
or iter(f) in [Trinder'91],
or ext(f) in [Tannen & Buneman'91].

For example,

$$\begin{aligned}
\text{filter}(p)x &= \text{hom}[set, set](\lambda a.\mathbf{if}\, p(a)\, \mathbf{then}\, \{a\}\, \mathbf{else}\, \{\})\,(x) \\
e \in x &= \text{hom}[set, some](\lambda a.(a = e))\,(x) \\
\text{length}(x) &= \text{hom}[list, sum](\lambda a.1)\,(x)
\end{aligned}$$

Restriction



C: Commutative

I: Idempotent

$\hom[T, S]$ is valid if and only if $T \preceq S$.

$$list \preceq bag \preceq set$$

Valid homomorphism:

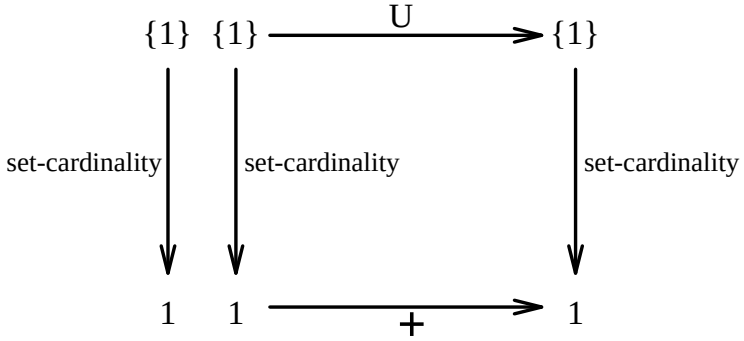
$$\text{bag-cardinality}(x) \;=\; \hom[bag, sum](\lambda a.1)\,(x)$$

since $bag \preceq sum$.

NOT a valid homomorphism:

$$\text{set-cardinality}(x) \;=\; \hom[set, sum](\lambda a.1)\,(x)$$

since $set \not\preceq sum$.



Monoid Comprehensions

For a monoid S:

$S\{\, e \mid r_1, \ldots, r_n \,\}$ where r_i is either $x_i \leftarrow e_i$ or $pred$

$$
\begin{array}{lcl}
S\{\, e \mid \,\} & \rightarrow & \text{unit}[S](e) \\
S\{\, e \mid x \leftarrow u,\ r_1, \ldots, r_n \,\} & \rightarrow & \text{hom}[T, S](\lambda x.S\{\, e \mid r_1, \ldots, r_n \,\})\,(u) \\
 & & \quad (\text{where } u : T \text{ and } T \preceq S) \\
S\{\, e \mid \text{pred},\ r_1, \ldots, r_n \,\} & \rightarrow & \textbf{if}\ \text{pred}\ \textbf{then}\ S\{\, e \mid r_1, \ldots, r_n \,\}\ \textbf{else}\ \text{zero}[S]
\end{array}
$$

Example:

$$
\begin{array}{rcl}
\text{join}_{pred}(x, y) & = & set\{\, (a, b) \mid a \leftarrow x,\ b \leftarrow y,\ pred \,\} \\
 & = & \text{hom}[set, set](\lambda a.\text{hom}[set, set](\lambda b.\textbf{if}\ pred\ \textbf{then}\ \{(a, b)\} \\
 & & \qquad \textbf{else}\ \{\})\,(y))\,(x)
\end{array}
$$

Examples from NF^2:

Unnesting:

$$\text{unnest}(x) \;=\; set\{\, e \mid s \leftarrow x,\ e \leftarrow s \,\}$$

Nesting:

$$\text{nest}(k)\,x \;=\; set\{\, \langle \text{key} : k(e), \text{partition} : set\{\, a \mid a \leftarrow x,\ k(e) = k(a) \,\}\rangle \mid e \leftarrow x \,\}$$

But you can also join a list with a bag and return a set:

$$set\{\,(a,b)\,|\,a \leftarrow [1,2,3],\ b \leftarrow \{\!\{4,5\}\!\}\,\}$$
$$= \{(1,4),(1,5),(2,4),(2,5),(3,4),(3,5)\}$$

Other examples:

$$\begin{array}{lcll}
\text{filter}(p)(x) & = & set\{\,e\,|\,e \leftarrow x,\ p(e)\,\} & x : set \\
x \cap y & = & set\{\,e\,|\,e \leftarrow x,\ e \in y\,\} & x, y : set \\
\text{length}(x) & = & sum\{\,1\,|\,e \leftarrow x\,\} & x : list \\
\text{sum}(x) & = & sum\{\,e\,|\,e \leftarrow x\,\} & x : bag \\
\text{count}(x,a) & = & sum\{\,1\,|\,e \leftarrow x,\ e = a\,\} & x : bag \\
a \in x & = & some\{\,a = e\,|\,e \leftarrow x\,\} & x : set \\
\exists a \in x : e & = & some\{\,e\,|\,a \leftarrow x\,\} & x : set \\
\forall a \in x : e & = & all\{\,e\,|\,a \leftarrow x\,\} & x : set
\end{array}$$

Translating OQL into the Calculus

select e
  from x_1 **in** $e_1, \ldots, x_n$ **in** e_n
  where $pred$
	$bag\{\, e \mid x_1 \leftarrow e_1, \ldots, x_n \leftarrow e_n, pred \,\}$

select distinct e
  from x_1 **in** $e_1, \ldots, x_n$ **in** e_n
  where $pred$
	$set\{\, e \mid x_1 \leftarrow e_1, \ldots, x_n \leftarrow e_n, pred \,\}$

e_1 **intersect** e_2	$set\{\, x \mid x \leftarrow e_1, x$ **in** $e_2 \,\}$
for all x **in** e : $pred$	$all\{\, pred \mid x \leftarrow e \,\}$
exists x **in** e : $pred$	$some\{\, pred \mid x \leftarrow e \,\}$
e_1 **in** e_2	$some\{\, x = e_1 \mid x \leftarrow e_2 \,\}$
count(e)	$sum\{\, 1 \mid x \leftarrow e \,\}$
sum(e)	$sum\{\, x \mid x \leftarrow e \,\}$
flatten(e)	$T\{\, x \mid s \leftarrow e, x \leftarrow s \,\}$ where $T \in \{set, bag, list\}$
group x **in** e **by** $(p_1 : e_1)$	$set\{\, \langle p_1 : e_1, \text{partition} : set\{\, a \mid a \leftarrow e, p_1(a) = p_1(x) \,\}\rangle \mid x \leftarrow e \,\}$

Normalization

Canonical forms:

1) $\text{hom}[T, S](f)\,(path)$
2) $\textbf{if}\, path\, \textbf{then}\, e_1\, \textbf{else}\, e_2$
3) $\textbf{if}\, (e_1 \otimes e_2)\, \textbf{then}\, e_3\, \textbf{else}\, e_4$ where $\otimes \in \{=, <, >, \leq, \geq\}$

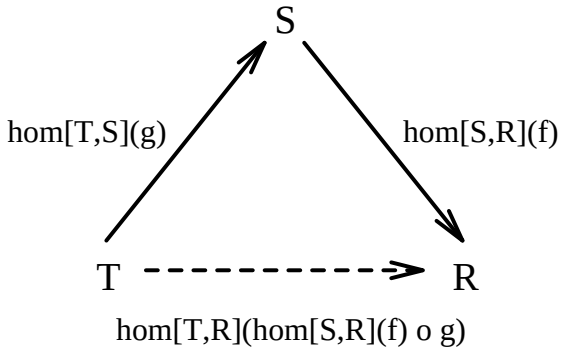
where

$$\begin{aligned} path &= var \\ path &= path.var \end{aligned}$$

Or equivalently [Wong'93]:

$$T\{\, e \mid \ldots,\, x \leftarrow path,\, \ldots,\, e_1 \otimes e_2,\, \ldots\,\}$$

Normalization rule:



$$\text{hom}[S, R](f)\,(\text{hom}[T, S](g)\,(e)) \;\rightarrow\; \text{hom}[T, R](\lambda x.\text{hom}[S, R](f)\,(g(x)))\,(e)$$

captures the transformation [Wong'93]:

$$T\{\, e_1 \mid a \leftarrow S\{\, e_2 \mid b \leftarrow x,\, \overline{r_1}\,\},\, \overline{r_2}\,\} \;\rightarrow\; T\{\, e_1 \mid b \leftarrow x,\, \overline{r_1},\, a \Leftarrow e_2,\, \overline{r_2}\,\}$$

Algebraic optimization:

$$\begin{aligned}
&\text{filter}(p)(\text{filter}(q)\,(x)) \\
&\quad\rightarrow\ \text{hom}[set, set](\lambda a.\mathbf{if}\ p(a)\ \mathbf{then}\ \{a\}\ \mathbf{else}\ \{\})\\
&\qquad\qquad (\text{hom}[set, set](\lambda a.\mathbf{if}\ p(a)\ \mathbf{then}\ \{a\}\ \mathbf{else}\ \{\})\,(x)) \\
&\quad\rightarrow\ \text{hom}[set, set](\lambda a.\mathbf{if}\ p(a)\ \mathbf{then}\ (\mathbf{if}\ q(a)\ \mathbf{then}\ \{a\}\mathbf{else}\ \{\})\ \mathbf{else}\ \{\})\,(x) \\
&\quad=\ \text{filter}(p \wedge q)\,x
\end{aligned}$$

Nested OQL query:

$$\begin{aligned}
&\mathbf{select}\ r \\
&\quad \mathbf{from}\ r\ \mathbf{in}\ R \\
&\quad \mathbf{where}\ r.B\ \mathbf{in}\ (\mathbf{select}\ s.D \\
&\qquad\qquad\qquad\qquad \mathbf{from}\ s\ \mathbf{in}\ S \\
&\qquad\qquad\qquad\qquad \mathbf{where}\ r.C = s.C)
\end{aligned}$$

$$\begin{aligned}
&=\ set\{\, r \mid r \leftarrow R,\ some\{\, x = r.B \mid x \leftarrow set\{\, s.D \mid s \leftarrow S,\ r.C = s.C \,\}\,\}\,\} \\
&\rightarrow\ set\{\, r \mid r \leftarrow R,\ s \leftarrow S,\ s.D = r.B,\ r.C = s.C \,\}
\end{aligned}$$

Why canonical forms are important?

- has been proved so for other domains (e.g. disjunctive normal forms);
- it is easier to prove general theorems about properties of programs in canonical form [Wong'93];
- normalization makes programs more efficient in *most* cases;
- the evaluation of canonical forms generates less intermediate data structures;
- normalized programs are amenable to a higher degree of parallelism.

Current extensions

The algebra can be easily extended to support:

- arrays with *real* matrix manipulations;
- object identity (yet another monoid);
- destructive database updates;
- non-deterministic operations (e.g. *listify*);
- pattern matching in comprehensions.