Language-Integrated Query with Nested Data Structures and Grouping

Rui Okura and Yukiyoshi Kameyama

FLOPS 2020 @ Online

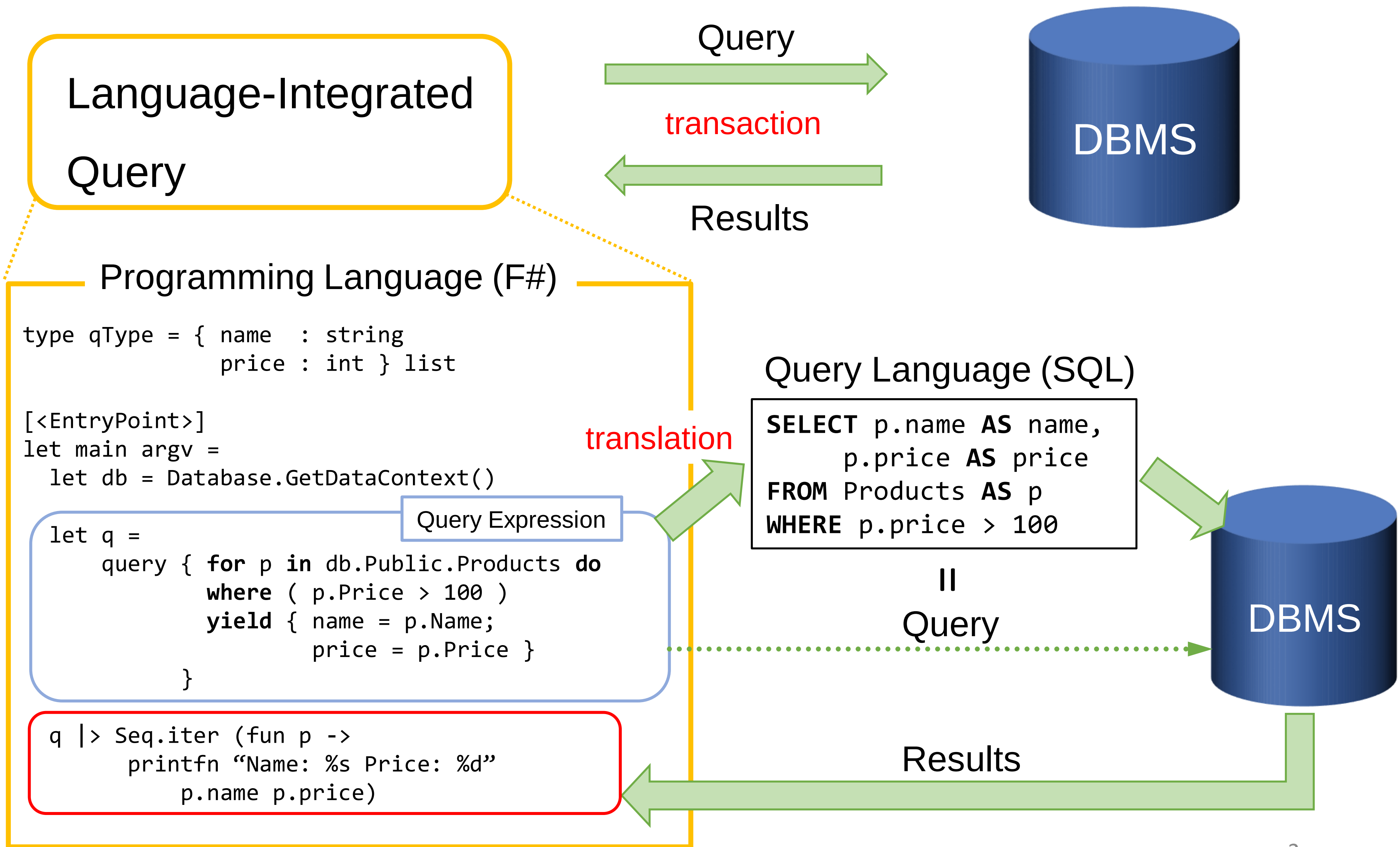
Sep. 2020

筑波大学
University of Tsukuba

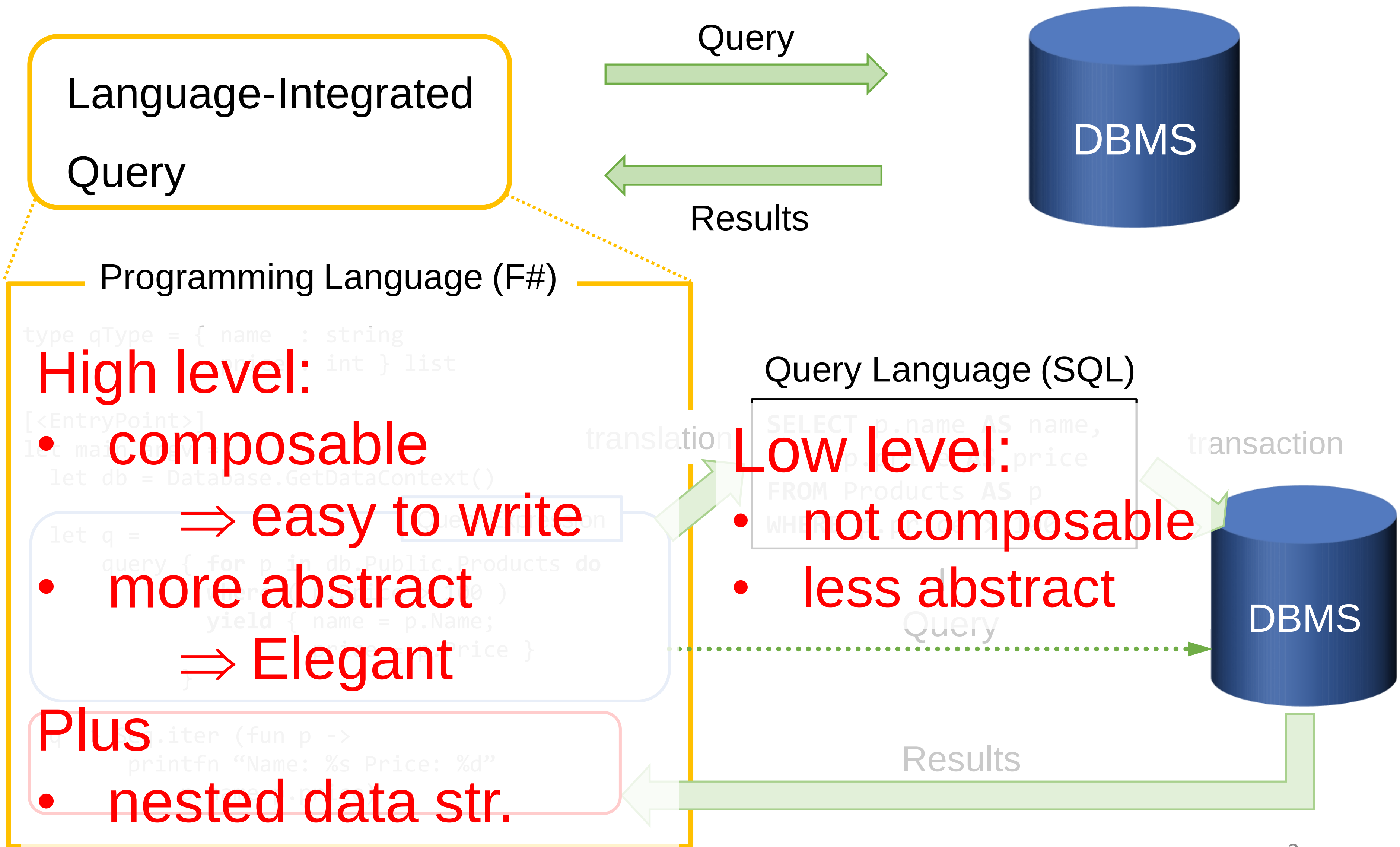
What is Language-Integrated Query ?

Language-Integrated Query with Nested Data Structures and Grouping

Language-Integrated Query



The Merit of Language-Integrated Query



High level:

- composable
  ⇒ easy to write
- more abstract
  ⇒ Elegant

Plus

- nested data str.

Low level:

- not composable
- less abstract

Language-Integrated Query
with Nested Data Structures
and Grouping

What are Nested Data Structures ?

Language-Integrated Query with Nested Data Structures

Programming Language (F#)

```
type QT1 = { pid : int
             order : { pid : int
                       sales : int } list } list
```

nested data str.

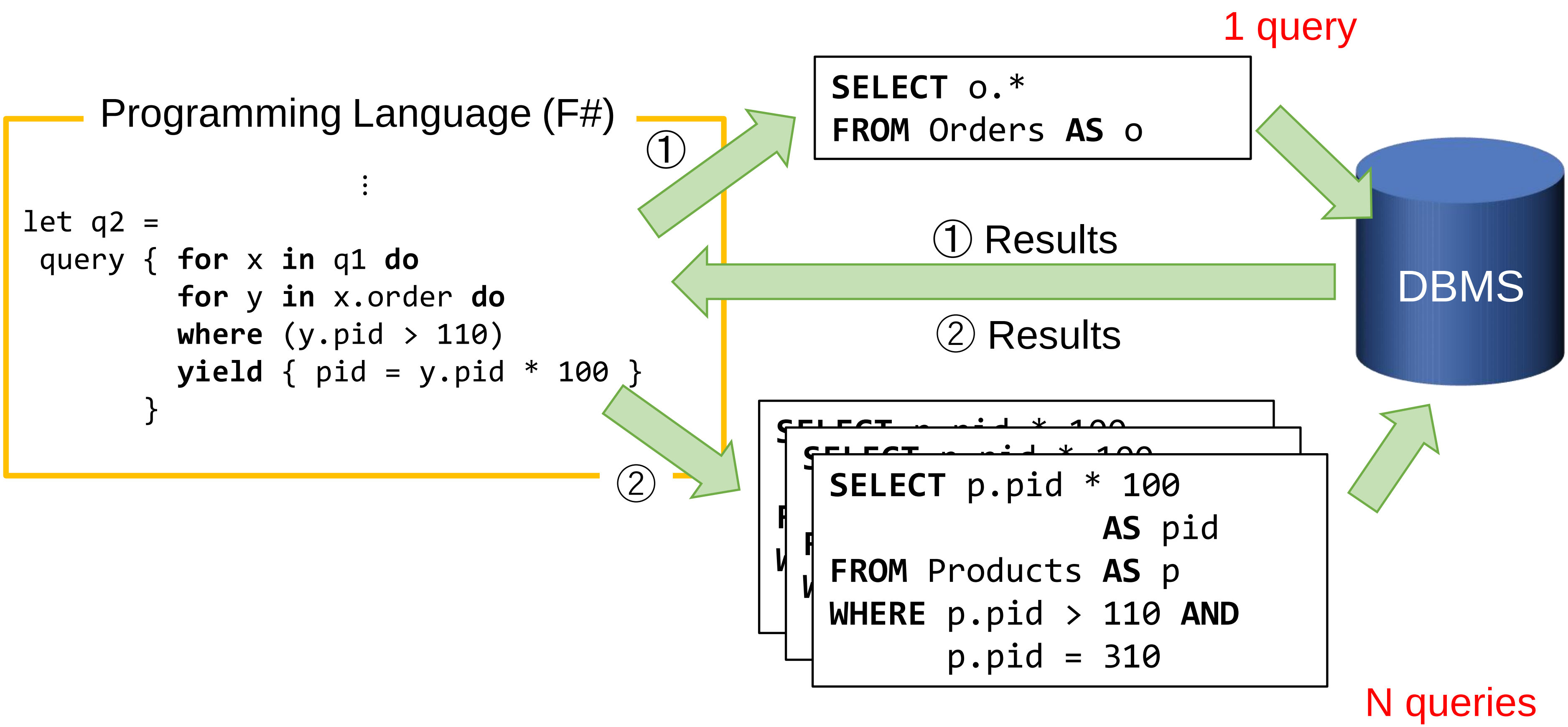
```
type QT2 = { pid : int } list
```

table type =
list of records of basic types

```
let q1 : QT1 = for o in db.Public.Orders do
               yield { pid = o.pid;
                       order = for p in db.Public.Products do
                               where ( p.pid = o.pid )
                               yield { pid = p.pid;
                                       sales = p.price * o.qty }
               }

let q2 : QT2 = query { for x in q1 do
                       for y in x.order do
                       where (y.pid > 110)
                       yield { pid = y.pid * 100 }
                     }
```

N+1 Query Problem



N+1 queries ….

Previous Work (1/2)

- Cooper 2009
  - Proposed rewriting rules for Nested Relational Calculus
  - Any closed term is **normalized** to a **non-nested** query, which is translated to a **single** SQL query.

- Cheney, Lindley, Wadler 2013
  - Formalized Cooper's idea in typed 2-level language T-LINQ
  - Example of rewriting rules:
    - **for** y **in** (**for** x **in** L **do** M) **do** N

      ↪ **for** x **in** L **do** (**for** y **in** M **do** N)

Previous Work (2/2)

- Cooper 2009
  - Proposed rewriting rules for Nested Relational Calculus (NRC) [Bunemann]
  - Any closed term is **normalized** to a **non-nested** query, which is translated to a **single** SQL query.
- Cheney, Lindley, Wadler 2013
  - Ex
    - **for** y **in** (**for** x **in** L **do** M) **do** N

      ↪ **for** x **in** L **do** (**for** y **in** M **do** N)

[Cooper 2009] [Cheney+ 2013]
"It is a future work / open problem to handle aggregation and grouping."

Language-Integrated Query with Nested Data Structures **and Grouping**

What are grouping in database queries ?

Grouping

Students table

name	dept	gpa
Suzuki	Math	4.0
Sato	CS	2.5
Takahashi	CS	3.8
Tanaka	Math	2.7
Ito	CS	3.5

Result table

dept	gpa_avg
Math	3.4
CS	3.3

```
SELECT s.dept AS dept,
       AVG(s.gpa) AS gpa_avg
FROM Students AS s
GROUP BY s.dept
```

Grouping
= **GROUP BY**
and **Aggregation** in SQL

E.g.)

SUM, AVG, MAX, MIN, COUNT

Handling Grouping

The N+1 problem for grouping has no solution in the original setting.

```
SELECT MAX(z.gpa_avg) AS result
FROM (SELECT s.class AS class,
             s.dept AS dept,
             AVG(s.gpa) AS gpa_avg
      FROM students AS s
      GROUP BY s.class, s.dept) AS z
GROUP BY z.dept
```

There's no single operation for taking AVG-MAX!

Fact: several important SQL allow **subqueries** (nested control structures)

e.g. PostgreSQL and MySQL

⇒ We can translate the above query to a single SQL query in such SQLs.

This work

Language-integrated query	Target SQL	N+1 query problem
without grouping and aggregation	**No nested control str.** **No nested data str.**	**Solved** [Cooper2009] [Cheney et al.2013]
with grouping and aggregation	**No nested control str.** **No nested data str.**	**No solution**
with grouping and aggregation	**Allows nested control str.** **No nested data str.**	**This work**

The N+1 query problem: Given a closed query of a table type, can we always translate it to an equivalent, single SQL query?

Non-solution (naïve approach)

Introducing a primitive for grouping: $\textbf{gfor}\ (x \leftarrow L\ ;\ \text{key})\ N$

$$
\begin{aligned}
&\textbf{gfor}\ (x \leftarrow \textbf{for}(y \leftarrow \textbf{table}(t)) \\
&\qquad\qquad \textbf{yield}\ \{\, a = y.a; \\
&\qquad\qquad\qquad\quad b = \{\, c = y.b \,\}\,\}\ ;\ a) \\
&\textbf{yield}\ \{\, \text{result} = \text{SUM}(x.b.c) \,\}
\end{aligned}
$$

- No effective transformation for $\textbf{gfor}\ (x \leftarrow \textbf{for}(x \leftarrow L)\ M; K)\ N$
- Hence, nested data structure remains as intermediate data.

$\Rightarrow$ This query can not be translated to SQL.

※ $\textbf{gfor}\ (x \leftarrow \textbf{for}(x \leftarrow L)\ M; K)\ N$

$\quad\not\hookrightarrow\ \textbf{for}(x \leftarrow L)\ \textbf{gfor}\ (x \leftarrow M; K)\ N$

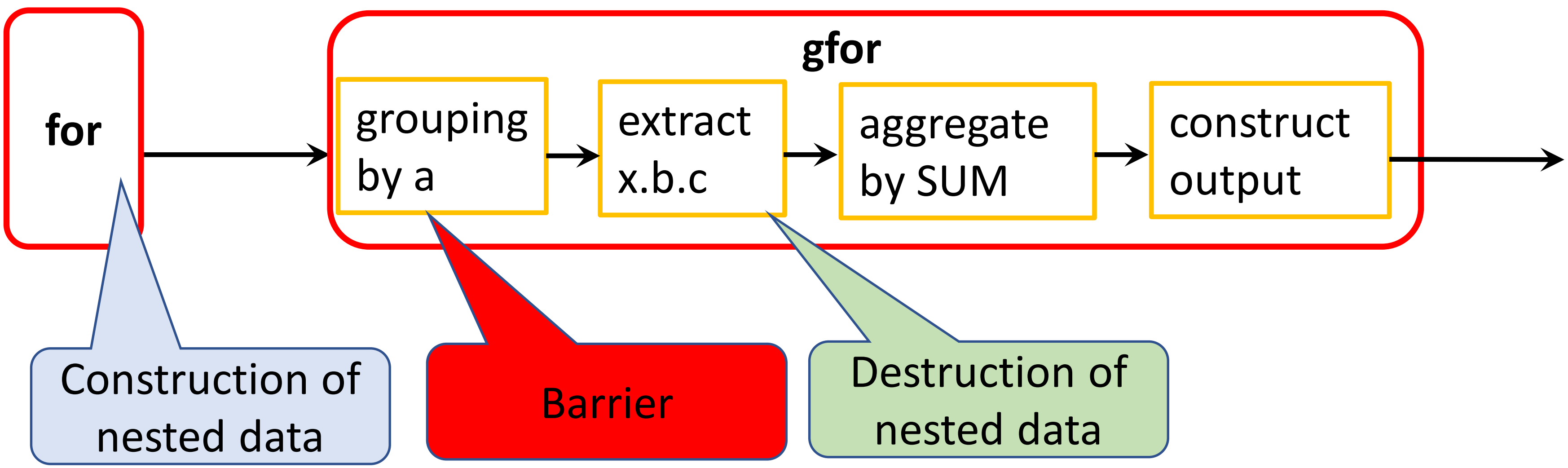
Hint from the naïve approach

$$
\begin{aligned}
&\textbf{gfor}\ (x \leftarrow \textbf{for}(y \leftarrow \textbf{table}(t)) \\
&\qquad\qquad \textbf{yield}\ \{\, a = y.a; \\
&\qquad\qquad\qquad\quad b = \{\, c = y.b \,\} \,\}\, ; a) \\
&\textbf{yield}\ \{\, \text{result} = \text{SUM}(x.b.c) \,\}
\end{aligned}
$$

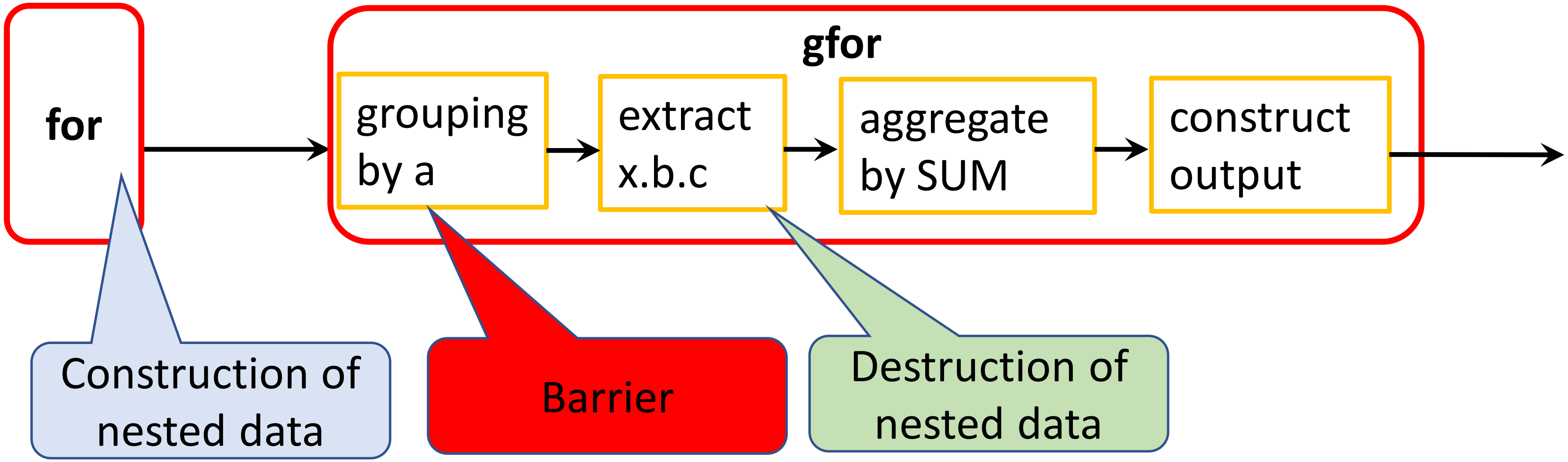
Construction of nested data

Destruction of nested data

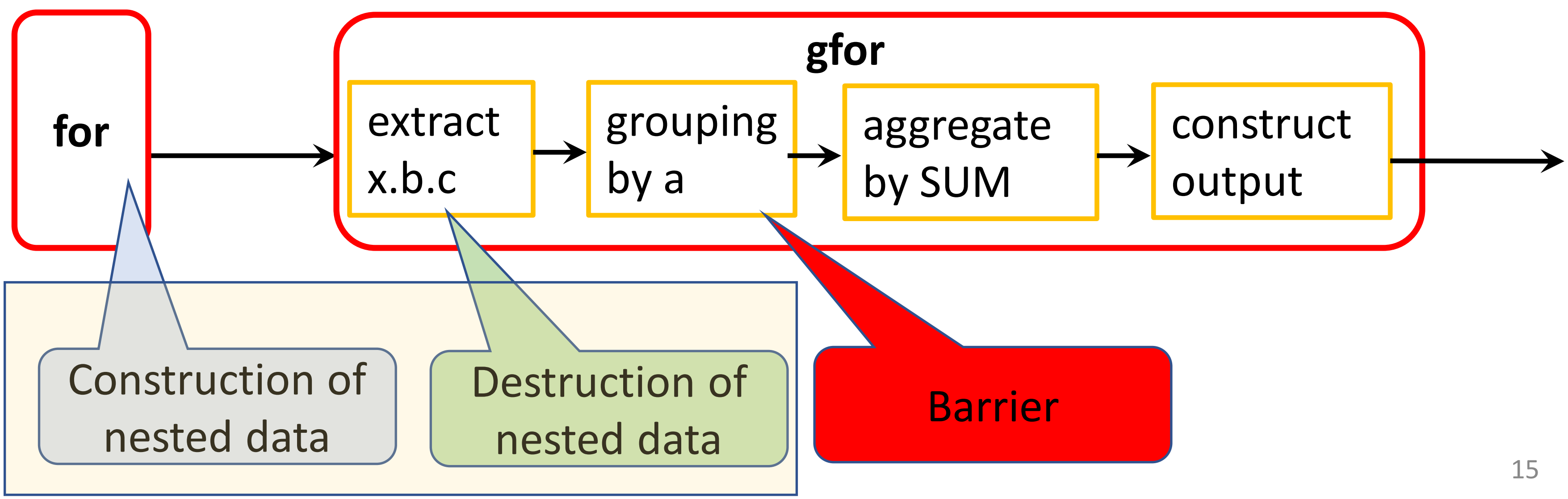
The construction-destruction pair of nested data should be eliminated.
It is not eliminated because the gfor primitive is a barrier between the two.



Hint from the naïve approach



We can swap the first two operations (their order does not matter).



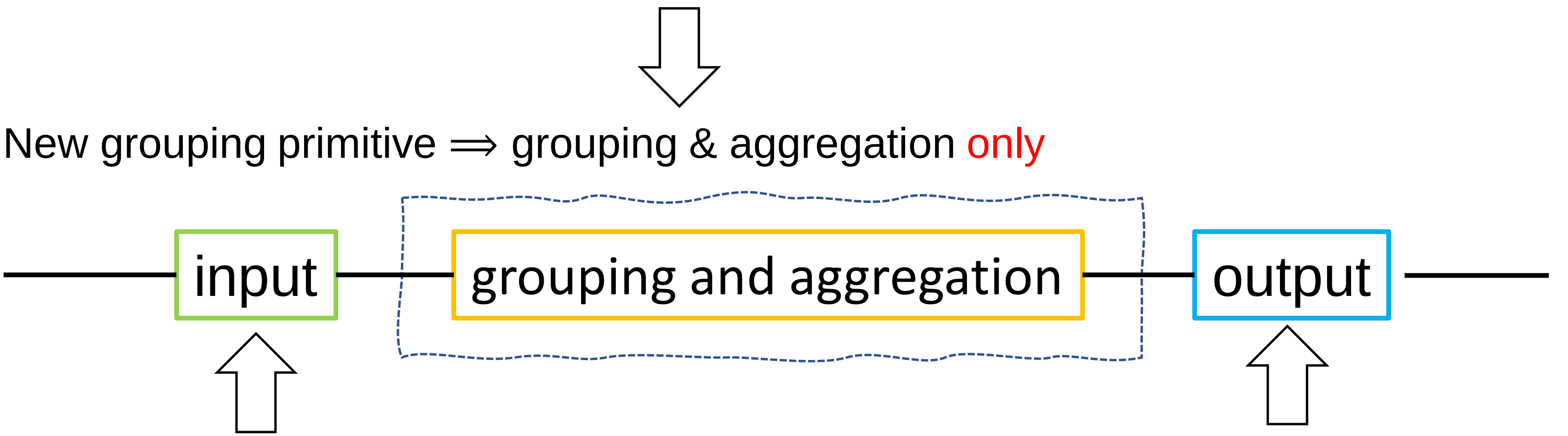
We can eliminate the construction-destruction pair.

Key idea (1/2)

- Decomposing **GROUP BY into small pieces:**

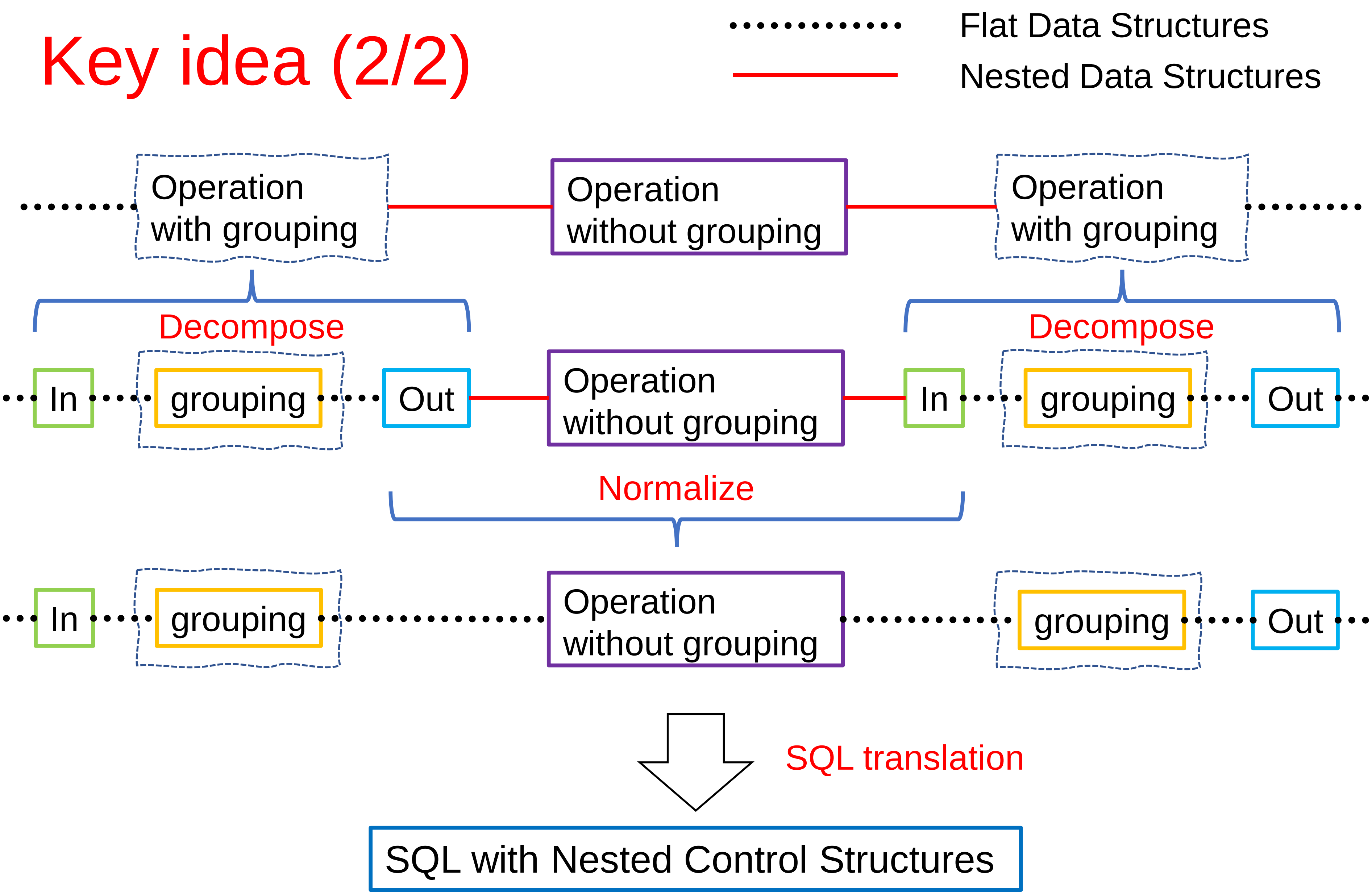
GROUP BY ⟹ input + grouping & aggregation part + output part

New grouping primitive ⟹ grouping & aggregation only



These parts can be expressed by the existing primitives

Key idea (2/2)


Flat Data Structures
Nested Data Structures
Operation
with grouping
Operation
without grouping
Operation
with grouping
Decompose
Decompose
In
grouping
Out
Operation
without grouping
In
grouping
Out
Normalize
In
grouping
Operation
without grouping
grouping
Out
SQL translation
SQL with Nested Control Structures

Quel : Base Language

Typing rule of **for**-operator

$$\frac{\Gamma \vdash M : \text{Bag } A \quad \Gamma,\ x : A \ \vdash N : \text{Bag } B}{\Gamma \vdash \textbf{for}(x \leftarrow M)\ N : \text{Bag } B}$$

Types

Base types $O ::= Int \mid String \mid Bool$

Types $A ::= O \mid A \to B \mid \text{Bag } A \mid \{\overline{l : A}\}$

Syntax

Terms $L, M, N ::= \lambda x. M \mid M\ N \mid \oplus(\overline{M}) \mid x \mid c \mid \textbf{for}(x \leftarrow M)\ N \mid \textbf{where}\ L\ M$

$\mid \textbf{yield}\ M \mid [\,] \mid \textbf{exists}\ M \mid \textbf{table}(t) \mid \{\overline{l = M}\} \mid L.l$

E.g.)

for$(x \leftarrow$ **table**$(M))$
where L
yield N

translation

```
SELECT N
FROM M AS x
WHERE L
```

Quelg : Quel + Aggregation + Grouping

Syntax

Terms $L, M, N ::= \ \dots \mid \mathcal{G}_{(\kappa,\alpha)}(L)$

A-Spec $\alpha ::= \{\overline{(l,\ \circledcirc,\ l')}\}$

Typing rule of $\mathcal{G}$-operator

GROUPING

$$\frac{\Gamma \vdash L : \text{Bag}\ \{\overline{\kappa_i : O_i}, \overline{l_i : O'_i}\} \quad \kappa = \{\overline{l_i}\} \quad \alpha = \{(\overline{l_i, \circledcirc_i, l'_i})\} \quad \circledcirc_i : \text{Bag}\ O'_i \to O'_i}{\Gamma \vdash \mathcal{G}_{(\kappa,\alpha)}(L) : \text{Bag}\ \{\overline{\kappa_i : O_i}, \overline{l'_i : O'_i}\}}$$

E.g.)

$\mathcal{G}_{(\kappa,\alpha)}$ (**table**(M))
where $\alpha = \{(l, \circledcirc, l')\}$

translation

```
SELECT ⊚(y.l) AS l'
FROM M AS y
GROUP BY y.k
```

Quelg : Quel + Aggregation + Grouping

We proved the following theorem.

Theorem.

1. Normalization rules for Quelg preserve typing, namely, $\Gamma \vdash L : A$ and $L \rightsquigarrow N$, then $\Gamma \vdash M : A$. (Subject Reduction)

2. For any typable term, normalization weakly terminates, namely, if $\Gamma \vdash L : A$, then there is a normal form N such that $L \rightsquigarrow N$. (Weak Normalization)

3. Suppose N is a normal form, $\cdot \vdash N : F$ is derivable where F is a table type. Then its type derivation contains only subtypes of table types. (Subformula Property)

Example of normalization (1/2)

gfor $(x \leftarrow$ **for**$(y \leftarrow$ **table**$(t))$
yield $\{\, a = y.a;$
$b = \{\, c = y.b \,\}\,\}\,;\, a)$
yield $\{\, \text{result} = \text{SUM}(x.b.c) \,\}$

$\mathcal{G}_{(f,\,\alpha)}$(**for**$(x \leftarrow$ **for**$(y \leftarrow$ **table**$(t))$
yield $\{\, a = y.a;$
$b = \{\, c = y.b \,\}\,\}\,)$
yield $\{\, f = x.a, \quad g = x.b.c \,\}\,)$
where $\alpha = \{(\, g,\ \text{SUM}, \text{result})\}$

$\mathcal{G}_{(f,\,\alpha)}$(**for**$(y \leftarrow$ **table**$(t))$
for$(x \leftarrow$ **yield** $\{\, a = y.a;$
$b = \{\, c = y.b \,\}\,\}\,)$
yield $\{\, f = x.a, \quad g = x.b \,\}\,)$
where $\alpha = \{(\, g,\ \text{SUM}, \text{result})\}$

Example of normalization (2/2)

$$\mathcal{G}_{(f,\alpha)}(\textbf{for}(y \leftarrow \textbf{table}(t))$$
$$\quad \textbf{for}(x \leftarrow \textbf{yield}\ \{\, a = y.a;$$
$$\qquad\qquad b = \{\, c = y.b \,\}\,\})$$
$$\quad \textbf{yield}\ \{\, f = x.a, \quad g = x.b.c \,\}\,)$$
where $\alpha = \{(\, g,\ \mathrm{SUM}, \mathrm{result})\}$

$$\mathcal{G}_{(f,\alpha)}(\textbf{for}(y \leftarrow \textbf{table}(t))$$
$$\quad \textbf{yield}\ \{\, f = \{\, a = y.a;\ b = \{\, c = y.b \,\}\,\}.a,$$
$$\qquad g = \{\, a = y.a;\ \ b = \{\, c = y.b \,\}\,\}.b.c \,\}\,)$$
where $\alpha = \{(\, g,\ \mathrm{SUM}, \mathrm{result})\}$

$$\mathcal{G}_{(f,\alpha)}(\textbf{for}(y \leftarrow \textbf{table}(t))$$
$$\quad \textbf{yield}\ \{\, f = y.a;$$
$$\qquad g = y.b \,\}\,)$$
where $\alpha = \{(\, g,\ \mathrm{SUM}, \mathrm{result})\}$

```
SELECT SUM(x.g) AS result
FROM (SELECT y.a AS f,
             y.b AS g
      FROM t AS y) AS x
GROUP BY x.f
```

Implementation & Experiments

- We have extended Suzuki et al.’s tagless-final implementation for Quel to Quelg:
  - Tagless final = type-preserving embedding of DSL [Kiselyov et al.]
- We have conducted experiments with quries:
  - Queries with nested grouping and aggregation
  - Generated SQL queries < 15 lines
  - We have compared the performance of ours with Microsoft’s LINQ in F#.

Q5 in the Experiment

$$Q_5' = \lambda\, p.\ \mathcal{G}_{(\text{oid},\, \alpha)}\ (\textbf{for}(o \leftarrow \textbf{table}(\text{“orders”}))\ \textbf{where}\ p(o.\text{qty})\ \textbf{yield}\ o)$$

where $\alpha = \{(\text{qty}, \text{COUNT}, \text{qty\_count})\}$

$$Q_5 = Q_5'\ (\lambda\, x.\ x > 2)$$

```
SELECT x.oid AS oid, COUNT(x.qty) AS qty_count
FROM (SELECT o.*
      FROM orders AS o
      WHERE o.qty > 2) AS x
GROUP BY x.oid
```

Performance

	Quelg		LINQ (F#)	
	SQL generation time	Execution time of SQL	Execution time of SQL	Depth
Q1	0.029 ms	16.186 ms	14.781 ms	1
Q2	0.101 ms	14.067 ms	15.129 ms	2
Q3	1.148 ms	4.356 ms	Not Available	3
Q4	0.074 ms	0.952 ms	1.787 ms	3
Q5	0.196 ms	7.409 ms	Not Available	3
Q6	0.427 ms	14.143 ms	Not Available	2
Q7	0.043 ms	11.255 ms	9.556 ms	4
Q8	5.590 ms	18.041 ms	20.690 ms	4
Q9	9.310 ms	3732.620 ms	Avalanche	4

the number of rows of each tables is 10000

- SQL generation time : time for normalization and SQL translation.
- Execution time of SQL : time to get results for generated SQL.
- Depth : the number of subqueries.

Summary

- We have designed an extension of simplified T-LINQ which is:
  - Able to express **GROUP BY** as macros.
  - Able to normalize to a query without Nested Data Structures.
  - Thus solving the **GROUP BY** problem in the presence of **Nested Control Structures** (but no **Nested Data Structures**).

- Our result is close to Wong and Libkin's theoretical work on classic database theory, but we are more practical.