Query Compilation Based on the Flattening Transformation

Alex Ulrich

Universität Tübingen

Dagstuhl Seminar 14511

Rich Query Languages

$$
\begin{array}{lcl}
e & := & c \mid x \mid \texttt{table}(n) \mid \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \\
& \mid & p\ e_1 \ldots e_2 \mid \texttt{let } x = e_1 \texttt{ in } e_2 \\
& \mid & [\ e \mid q_1, \ldots, q_n\ ] \\
\\
q & := & x \leftarrow e \mid e \\
\\
p & := & \texttt{SUM} \mid \texttt{MIN} \mid \texttt{LENGTH} \mid \texttt{AND} \mid \ldots \\
& \mid & \texttt{SORT} \mid \texttt{NUMBER} \mid \texttt{APPEND} \mid \texttt{CONCAT} \mid \texttt{NULL} \mid \texttt{GROUP} \\
& \mid & \texttt{NUB} \mid \texttt{TAKE} \mid \texttt{DROP} \mid \texttt{ZIP} \mid \texttt{ELEM} \mid \ldots \\
& \mid & (e_1, \ldots, e_n) \mid [e] \mid \circledast \mid \bullet.i
\end{array}
$$

Looks Like Haskell – Database Supported Haskell (DSH)

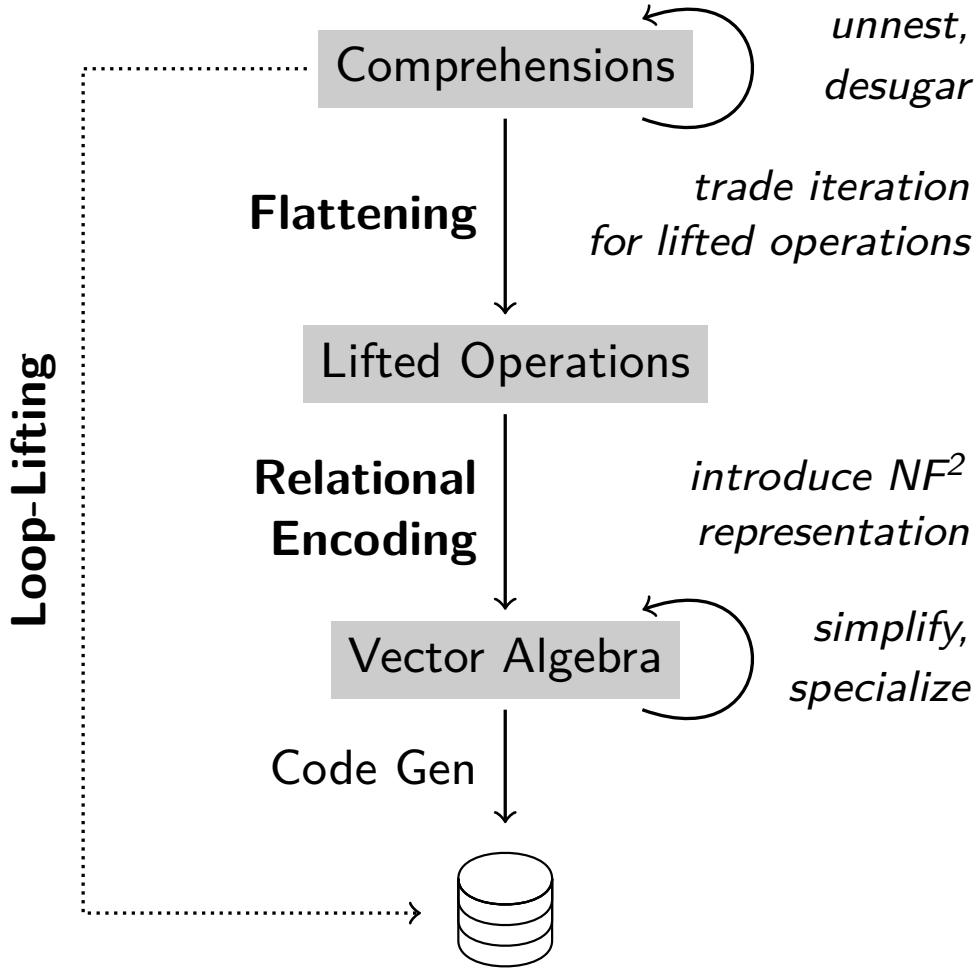
```
-- is customer c a resident of nation?
hasNationality :: Q Customer -> Text -> Q Bool
hasNationality c nation =
  or [ n_name n == toQ nation && n_nationkey n == c_nationkey c |
       n <- nations ]

-- all orders of customer c with the given status (O, P, C)
ordersWithStatus :: Text -> Q Customer -> Q [Order]
ordersWithStatus status c =
  [ o | o <- ordersOf c, o_orderstatus o == toQ status ]

-- our revenue for order o
revenue :: Q Order -> Q Double
revenue o = sum [ l_extendedprice l * (1 - l_discount l) |
                  l <- lineitems, l_orderkey l == o_orderkey o ]

-- expected revenues (by customer, with details) in nation
expectedRevenue :: Text -> Q [(Text, [(Date, Double)])]
expectedRevenue nation =
  [ (c_name c, [ (o_orderdate o, revenue o) |
                 o <- ordersWithStatus "P" c ]) |
    c <- customers,
    c `hasNationality` nation ]
```

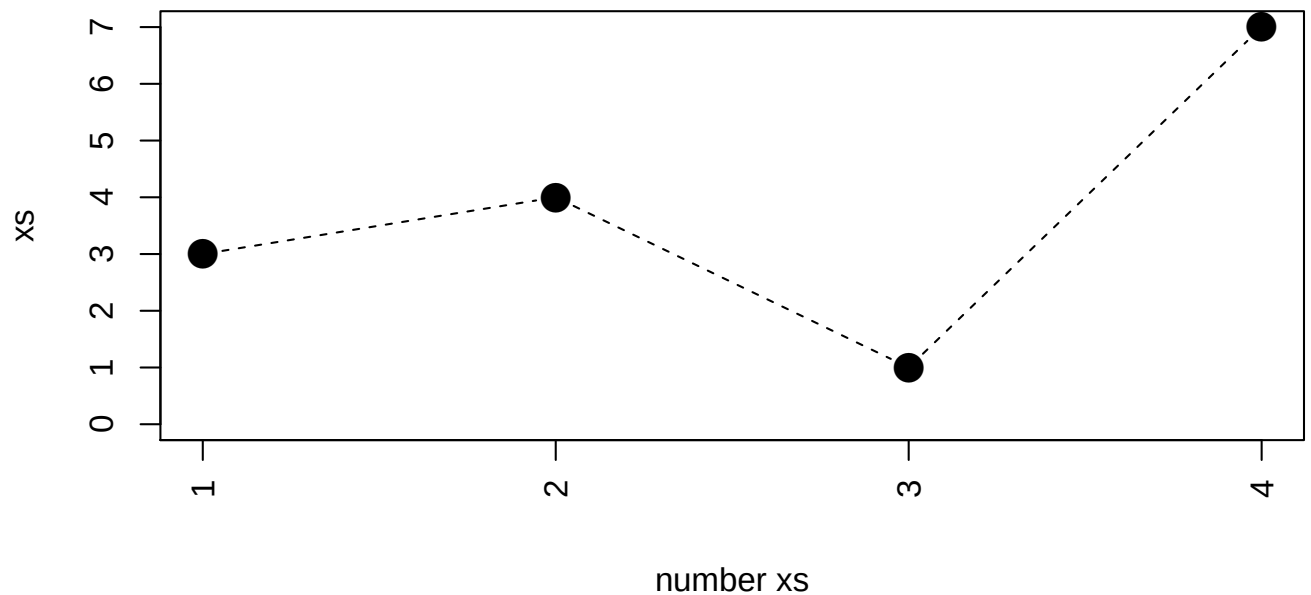
Compiler: From Monolithic to Small Steps



Nested Iteration

```
NUMBER [3,4,1,7] ≡ [(3, 1), (4, 2), (1, 3), (7, 4)]

[ AND [ y <= x | (y, j) <- NUMBER xs, j <= i ]
| (x, i) <- NUMBER xs
]
```



Comprehension Optimization (1990's)

- Optimization of monoid comprehensions, complex object queries (Buneman, Fegaras, Grust, Steenhagen, ...)
- List-based join operators

```
THETAJOIN{p} xs ys ≡ [ (x, y) | x <- xs, y <- ys, p x y ]
SEMIJOIN{p} xs ys ≡ [ x | x <- xs, OR [ p x y | y <- ys ] ]
ANTIJOIN{p} xs ys ≡ [ x | x <- xs, AND [ NOT (p x y) | y <- ys ] ]
NESTJOIN{p} xs ys ≡ [ (x, [ y | y <- ys, p x y ]) | x <- xs ]
```

- Example:

```
[ AND [ y <= x | (y, j) <- g ]
| ((x, i), g)
      <- NESTJOIN{Ⓛ.2 <= Ⓡ.2} (NUMBER xs) (NUMBER xs)
]
```

Flattening Transformation (Blelloch, 1990's)

- Explicit nested iteration...

```
[ AND [ y <= x | (y, j) <- g ]
| ((x, i), g)
      <- NESTJOIN{Ⓛ.2 <= Ⓡ.2} (NUMBER xs) (NUMBER xs)
]
```

Flattening Transformation (Blelloch, 1990's)

- Explicit nested iteration...

```
[ AND [ y <= x | (y, j) <- g ]
| ((x, i), g)
      <- NESTJOIN{Ⓛ.2 <= Ⓡ.2} (NUMBER xs) (NUMBER xs)
]
```

- ...replaced by lifted operators:

```
LET xg = NESTJOIN{Ⓛ.2 <= Ⓡ.2} (NUMBER xs) (NUMBER xs)
IN  AND¹ (xg.2¹.1² <=² (DIST¹ xg.1¹ xg.2¹))
```

Flattening Transformation (Blelloch, 1990's)

- Explicit nested iteration...

```
[ AND [ y <= x | (y, j) <- g ]
| ((x, i), g)
      <- NESTJOIN{Ⓛ.2 <= Ⓡ.2} (NUMBER xs) (NUMBER xs)
]
```

- ...replaced by lifted operators:

```
LET xg = NESTJOIN{Ⓛ.2 <= Ⓡ.2} (NUMBER xs) (NUMBER xs)
IN  AND¹ (xg.2¹.1² <=² (DIST¹ xg.1¹ xg.2¹))
```

- Lifted operators:

```
+  :: Int -> Int -> Int
+¹ :: [Int] -> [Int] -> [Int]
+² :: [[Int]] -> [[Int]] -> [[Int]]
...
```

Lifted Operators For Free

$+^1$ :: [Int] -> [Int] -> [Int]

Lifted Operators For Free

$+^1$ :: [Int] -> [Int] -> [Int]

$+^d$ :: [...[Int]...] -> [...[Int]...] -> [...[Int]...]

Lifted Operators For Free

$+^1$ :: [Int] -> [Int] -> [Int]

$+^d$ :: $\underbrace{[\ldots[}_{d-1}$[Int]]...] -> [...[[Int]]...] -> [...[[Int]]...]

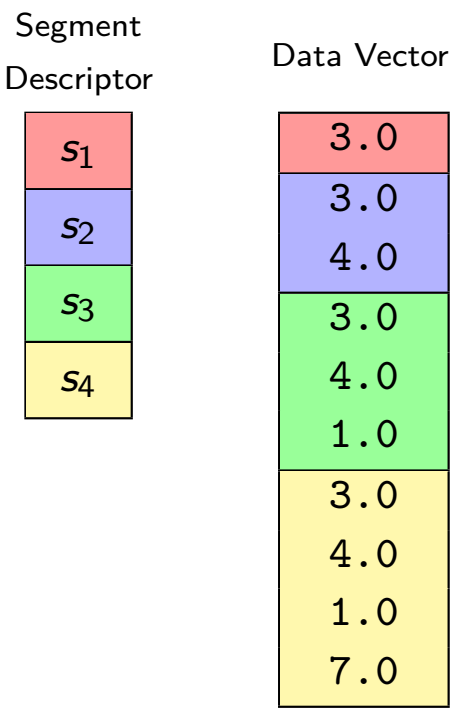
Lifted Operators For Free

$$+^1 \texttt{ :: [Int] -> [Int] -> [Int]}$$
$$+^d \texttt{ :: } \underbrace{\texttt{[...[}}_{d-1}\texttt{[Int]]...] -> [...[[Int]]...] -> [...[[Int]]...]}$$

$$\texttt{xs } +^d \texttt{ ys} \equiv \texttt{imprint}_{d-1}\texttt{ xs ((forget}_{d-1}\texttt{ xs) } +^1 \texttt{ (forget}_{d-1}\texttt{ ys))}$$

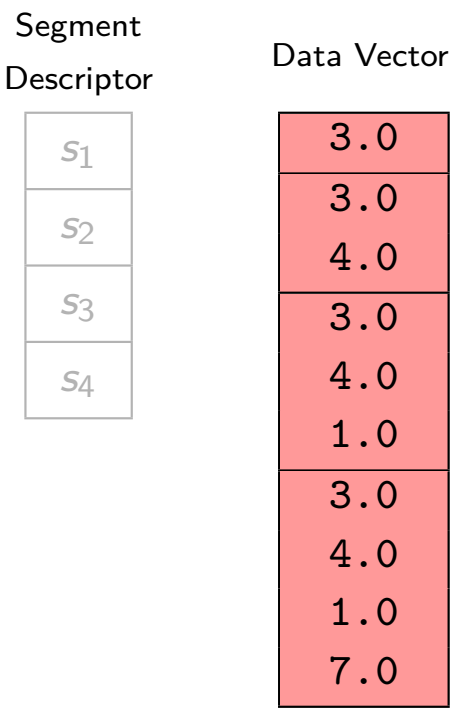
Separate Structure and Content

```
xs = [[3.0], [3.0,4.0], [3.0,4.0,1.0], [3.0,4.0,1.0,7.0]]
```



Separate Structure and Content

```
xs = [[3.0], [3.0,4.0], [3.0,4.0,1.0], [3.0,4.0,1.0,7.0]]
```



Separate Structure and Content

```
xs = [[3.0], [3.0,4.0], [3.0,4.0,1.0], [3.0,4.0,1.0,7.0]]
```

Segment Descriptor

s_1
s_2
s_3
s_4

Data Vector

3.0
3.0
4.0
3.0
4.0
1.0
3.0
4.0
1.0
7.0

Relational NF^2 Encoding

seg	pos
1	1
1	2
1	3
1	4

seg	pos	item
1	1	3.0
2	2	3.0
2	3	4.0
3	4	3.0
3	5	4.0
3	6	1.0
4	7	3.0
4	8	4.0
4	9	1.0
4	10	7.0

Separate Structure and Content

```
xs = [[3.0], [3.0,4.0], [3.0,4.0,1.0], [3.0,4.0,1.0,7.0]]
```

Segment Descriptor

s_1
s_2
s_3
s_4

Data Vector

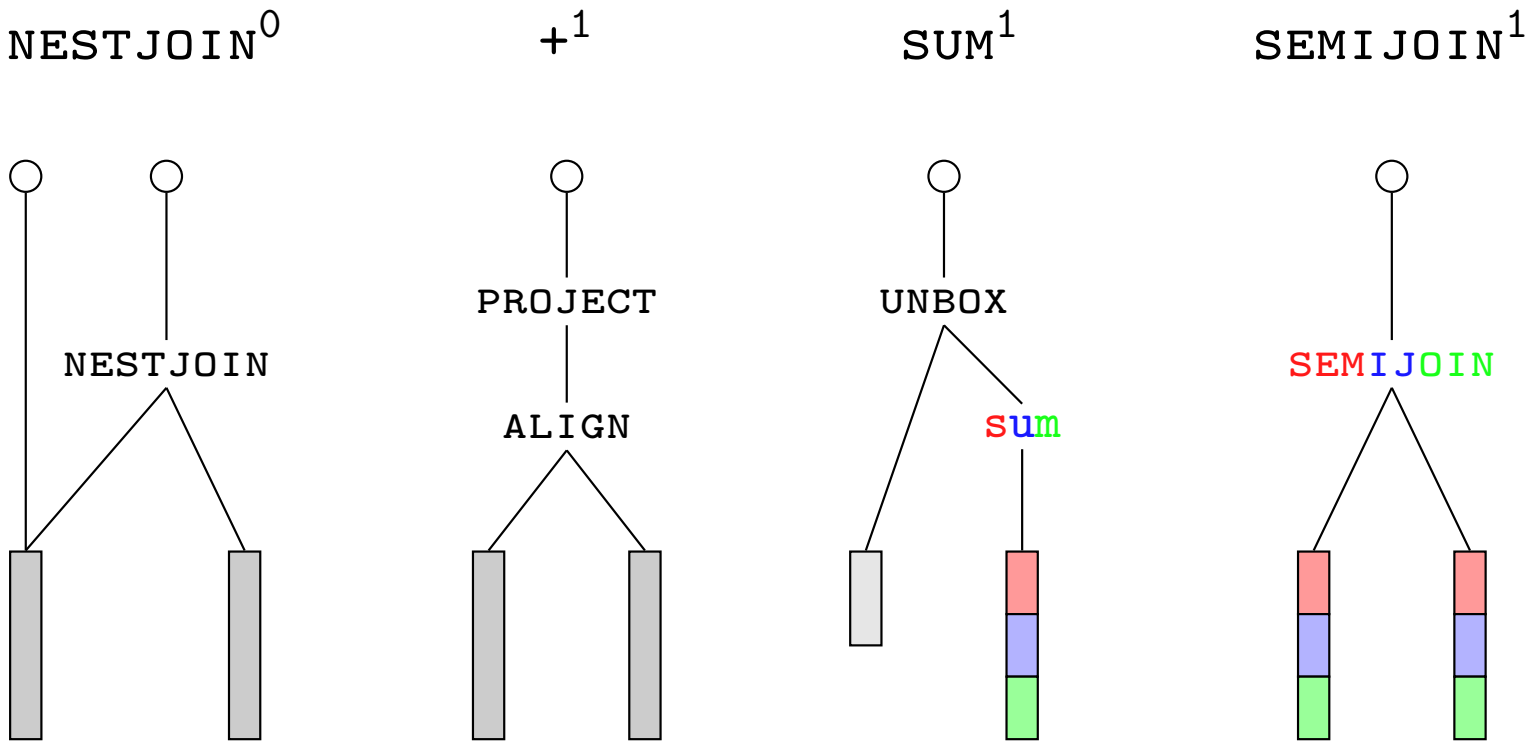
3.0
3.0
4.0
3.0
4.0
1.0
3.0
4.0
1.0
7.0

Relational NF^2 Encoding

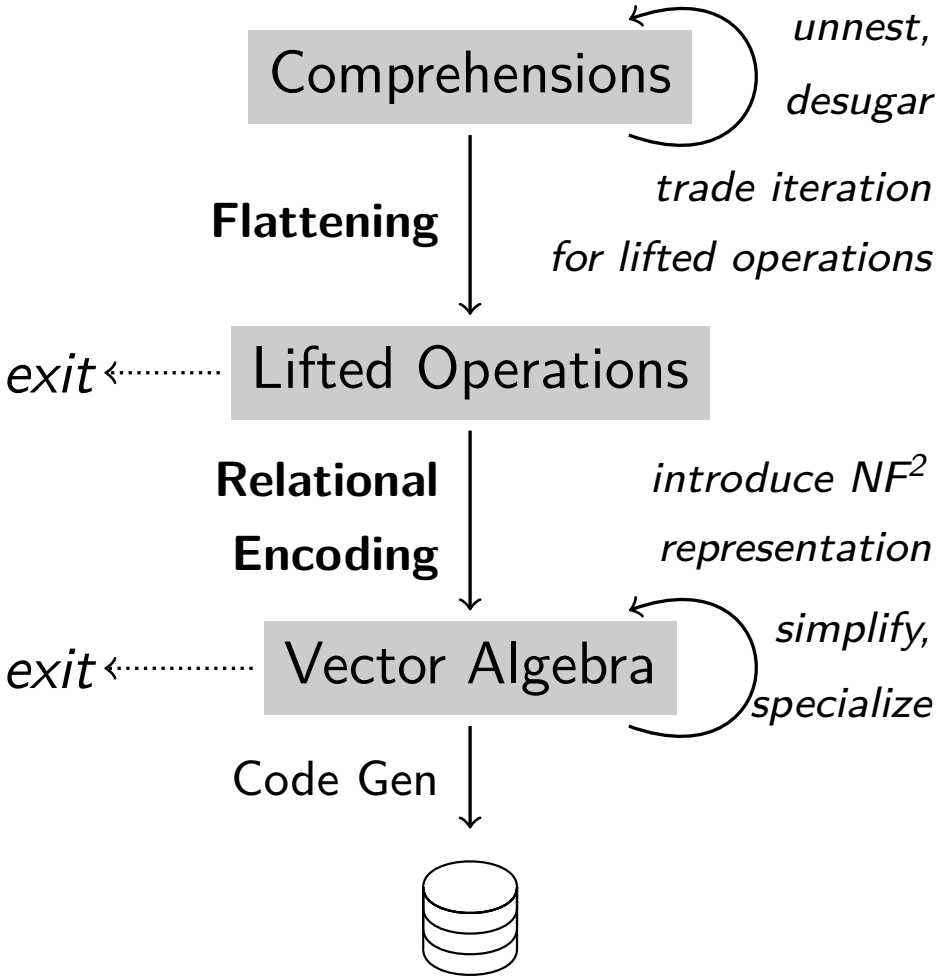
seg	pos
1	1
1	2
1	3
1	4

seg	pos	item
1	1	3.0
2	2	3.0
2	3	4.0
3	4	3.0
3	5	4.0
3	6	1.0
4	7	3.0
4	8	4.0
4	9	1.0
4	10	7.0

Lifted $\neq$ Fancy



Open Questions



- Flattening for unordered collections (multisets)?
- Implement flat vectors on non-relational query engines (array databases, Apache Flink, . . .)?
- Other data models that separate content from structure?