Flattening Nested Database Queries

Ben Lippmeier
University of New South Wales
FP-SYD 2014/05/28

Nested array representation

```
arr    = [[1 2] [3 4 5] [] [6]]

segd   = [2 3 0 1]
data   = [1 2 3 4 5 6]
```

Flat vs Nested Data Parallelism

- Flat Parallelism: Worker function is sequential.

```
thingo xs
  = mapP (\x. x + 1) xs
```

- Nested Parallelism: Worker function is parallel.

```
thingo xss
  = mapP (\xs. zipWithP g xs ys) xss
```

- The Flattening / Vectorisation transform converts nested parallelism into flat parallelism.

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys
```

```
fL :: Array Int -> Array Int
fL xs = xs +L (replicate n 1)
 where n = length xs

g :: Array Int -> Array Int
g ys = fL ys
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys
```

```
fL :: Array Int -> Array Int
fL xs = zipWithP (+) xs (replicate n 1)
 where n = length xs

g :: Array Int -> Array Int
g ys = fL ys
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys

h :: Array (Array Int) -> Array (Array Int)
h zss = mapP g zss
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys

h :: Array (Array Int) -> Array (Array Int)
h zss = mapP g zss
```

```
g :: Array Int -> Array Int
g ys = fL ys

gL :: Array (Array Int) -> Array (Array Int)
gL yss = fLL yss
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys

h :: Array (Array Int) -> Array (Array Int)
h zss = mapP g zss
```

```
g :: Array Int -> Array Int
g ys = fL ys

gL :: Array (Array Int) -> Array (Array Int)
gL yss = unconcatP yss (fL (concatP yss))
```

Nested array representation

```
arr    = [[1 2] [3 4 5] [] [6]]

segd   = [2 3 0 1]
data   = [1 2 3 4 5 1]
```

Relational Algebra: Mother Tongue—XQuery: Fluent

Torsten Grust Jens Teubner
University of Konstanz
Department of Computer & Information Science
Box D 188, 78457 Konstanz, Germany
{grust,teubner}@inf.uni-konstanz.de

ABSTRACT

This work may be seen as a further proof of the versatility of the relational database model. Here, we add XQuery to the catalog of languages which RDBMSs are able to "speak" fluently.

Given suitable relational encodings of sequences and ordered, unranked trees—the two data structures that form the backbone of the XML and XQuery data models—we describe a compiler that translates XQuery expressions into a simple and quite standard relational algebra which we expect to be efficiently implementable on top of any relational query engine. The compilation procedure is fully compositional and emits algebraic code that strictly adheres to the XQuery language semantics: document and sequence order as well as node identity are obeyed. We exercise special care in translating arbitrarily nested XQuery `FLWOR` iteration constructs into equi-joins, an operation which RDBMSs can perform particularly fast. The resulting purely relational XQuery processor shows promising performance figures in experiments.

types onto tables. Such encodings have also been proposed for *ordered, unranked trees*, the data type that forms the backbone of the XML data model. These mappings turn RDBMSs into *relational XML processors*. Furthermore, if the tree encoding is designed such that core operations on trees—XPath axis traversals—lead to efficient table operations, this can result in high-performance *relational XPath* implementations [8, 10].

In this work we extend the relational XML processing stack and propose the fully relational evaluation of XQuery [1] expressions. We give a compositional set of translation rules that compile XQuery expressions into a standard, quite primitive relational algebra. We expect any relational query engine to be able to efficiently implement the operators of this algebra. The operators were, in fact, designed to match the capabilities of modern SQL-based relational database systems (*e.g.*, the row numbering operator ϱ exactly mirrors SQL:1999 OLAP ranking functionality) [9].

By design, we only have minimalistic assumptions on the underlying tree encoding, met by several XML encoding schemes [4, 13]. Our algebra can be easily modified to oper-

TDM'04, the first Twente Data Management Workshop on XML Databases and Information Retrieval, Enschede, The Netherlands

Nested XQueries

$$
s \left\{
\begin{array}{l}
\texttt{for } \$v_0 \texttt{ in (1,2) return} \\
\qquad s_0 \left\{
\begin{array}{l}
\texttt{(} \$v_0 \texttt{ ,} \\
\quad \texttt{for } \$v_{0\cdot 0} \texttt{ in (10,20) return} \\
\qquad s_{0\cdot 0} \left\{ \texttt{(}\$v_0 \texttt{,} \$v_{0\cdot 0}\texttt{)} \right. \\
\texttt{)}
\end{array}
\right.
\end{array}
\right.
$$

Nested XQueries

```
s { for $v0 in (1,2) return
       s0 { ( $v0 ,
              for $v0·0 in (10,20) return
                 s0·0 { ($v0,$v0·0)
            )
```

iter	*pos*	*item*
1	1	"1"
1	2	"10"
2	1	"1"
2	2	"20"
3	1	"2"
3	2	"10"
4	1	"2"
4	2	"20"

(a) Intermediate result in $s_{0 \cdot 0}$.

iter	*pos*	*item*
1	1	"1"
1	2	"10"
1	3	"1"
1	4	"20"
2	1	"2"
2	2	"10"
2	3	"2"
2	4	"20"

(b) Intermediate result in s_0.

iter	*pos*	*item*
1	1	"1"
1	2	"1"
1	3	"10"
1	4	"1"
1	5	"20"
1	6	"2"
1	7	"2"
1	8	"10"
1	9	"2"
1	10	"20"

(c) Final result in top-level scope.

Nested array representation

```
arr    = [[1 2] [3 4 5] [] [6]]

segd   = [2 3 0 1]
data   = [1 2 3 4 5 1]
```

Nested array representation

```
arr     = [[1 2] [3 4 5] [] [6]]

segd    = [2 3 0 1]
data    = [1 2 3 4 5 1]

segdA’ = [0 0 1 1 1 3]
segdB’ = [0 1 0 1 2 0]
```

$e ::=$	c	atomic constants
$\mid$	$\$v$	variables
$\mid$	(e,e)	sequence construction
$\mid$	$e/\alpha::n$	loc. step (axis α, node test n)
$\mid$	$\texttt{element}\ t\ \{e\}$	element constructor (tag t)
$\mid$	$\texttt{for}\ \$v\ \texttt{in}\ e\ \texttt{return}\ e$	iteration
$\mid$	$\texttt{let}\ \$v\ \texttt{:=}\ e\ \texttt{return}\ e$	let binding
$\mid$	$e+e$	addition

$e ::=$	c	atomic constants
$\mid$	`$`v	variables
$\mid$	`(`e`,`e`)`	sequence construction
$\mid$	e`/`α`::`n	loc. step (axis α, node test n)
$\mid$	**`element`** t `{`e`}`	element constructor (tag t)
$\mid$	**`for`** `$`v **`in`** e **`return`** e	iteration
$\mid$	**`let`** `$`v `:=` e **`return`** e	let binding
$\mid$	e`+`e	addition

$\pi_{a_1:b_1,\ldots,a_n:b_n}$	projection (and renaming)
σ_a	selection
$\dot{\cup}$	disjoint union
$\times$	cartesian product
$\bowtie_{a=b}$	equi-join
$\varrho_{b:\langle a_1,\ldots,a_n\rangle/p}$	row numbering
$⊿_{\alpha,n}$	XPath axis join (axis α, node test n)
ε	element construction
$\circledast_{b:\langle a_1,\ldots,a_n\rangle}$	n-ary arithmetic/comparison operator $*$
$\underline{a\,\vert\,b}$	literal table

$$\boxed{\Gamma; \mathsf{loop}; \Delta \vdash e \Mapsto (q, \Delta')}$$

$$\frac{}{\Gamma; \mathsf{loop}; \Delta \vdash c \Mapsto \left(\mathsf{loop} \times \begin{array}{c|c} pos & item \\ \hline 1 & c \end{array}, \Delta\right)} \; . \qquad \text{(Const)}$$

```
Ln[ c ] = replicate n c
```

$$\boxed{\Gamma; \mathsf{loop}; \Delta \vdash e \Mapsto (q, \Delta')}$$

$$\frac{\Gamma; \mathsf{loop}; \Delta \vdash e_1 \Mapsto (q_1, \Delta_1) \qquad \Gamma; \mathsf{loop}; \Delta_1 \vdash e_2 \Mapsto (q_2, \Delta_2)}{\Gamma; \mathsf{loop}; \Delta \vdash e_1 \;\mathsf{+}\; e_2 \Mapsto \left(\pi_{iter,pos,item:res} \left(\oplus_{res:\langle item,item' \rangle}(q_1 \Join_{iter=iter'} \left(\pi_{iter':iter,item':item} q_2\right)\right)\right), \Delta_2\right)}$$

$$\boxed{\Gamma; \mathsf{loop}; \Delta \vdash e \Mapsto (q, \Delta')}$$

$$\frac{\Gamma; \mathsf{loop}; \Delta \vdash e_1 \Mapsto (q_1, \Delta_1) \qquad \Gamma; \mathsf{loop}; \Delta_1 \vdash e_2 \Mapsto (q_2, \Delta_2)}{\Gamma; \mathsf{loop}; \Delta \vdash e_1 \text{ + } e_2 \Mapsto \left(\pi_{iter,pos,item:res}\left(\oplus_{res:\langle item,item' \rangle}(q_1 \bowtie_{iter=iter'} (\pi_{iter':iter,item':item} q_2))\right), \Delta_2\right)}$$

iter	*pos*	*item*
1	1	"1"
1	2	"10"
2	1	"1"
2	2	"20"
3	1	"2"
3	2	"10"
4	1	"2"
4	2	"20"

(a) Intermediate result in $s_{0.0}$.

iter	*pos*	*item*
1	1	"1"
1	2	"10"
1	3	"1"
1	4	"20"
2	1	"2"
2	2	"10"
2	3	"2"
2	4	"20"

(b) Intermediate result in s_0.

iter	*pos*	*item*
1	1	"1"
1	2	"1"
1	3	"10"
1	4	"1"
1	5	"20"
1	6	"2"
1	7	"2"
1	8	"10"
1	9	"2"
1	10	"20"

(c) Final result in top-level scope.

$$\Gamma; \mathsf{loop}; \Delta \vdash e \Mapsto (q, \Delta')$$

$$\frac{\Gamma; \mathsf{loop}; \Delta \vdash e_1 \Mapsto (q_1, \Delta_1) \qquad \Gamma; \mathsf{loop}; \Delta_1 \vdash e_2 \Mapsto (q_2, \Delta_2)}{\Gamma; \mathsf{loop}; \Delta \vdash e_1 \texttt{ + } e_2 \Mapsto \left(\pi_{iter,pos,item:res}\left(\oplus_{res:\langle item,item'\rangle}(q_1 \bowtie_{iter=iter'} (\pi_{iter':iter,item':item}\, q_2))\right), \Delta_2\right)}$$

iter	*pos*	*item*
1	1	"1"
1	2	"10"
2	1	"1"
2	2	"20"
3	1	"2"
3	2	"10"
4	1	"2"
4	2	"20"

(a) Intermediate result in $s_{0.0}$.

iter	*pos*	*item*
1	1	"1"
1	2	"10"
1	3	"1"
1	4	"20"
2	1	"2"
2	2	"10"
2	3	"2"
2	4	"20"

(b) Intermediate result in s_0.

iter	*pos*	*item*
1	1	"1"
1	2	"1"
1	3	"10"
1	4	"1"
1	5	"20"
1	6	"2"
1	7	"2"
1	8	"10"
1	9	"2"
1	10	"20"

(c) Final result in top-level scope.

$$L_n[\ e_1 + e_2\ ] = Ln[\ e_1\ ] +L\ Ln[\ e_2\ ]$$

$$\dfrac{\begin{array}{ll}\{\ldots, \$v_i \mapsto q_{v_i}, \ldots\}; \mathsf{loop}; \Delta \vdash e_1 \mapsto (q_1, \Delta_1) & q_v \equiv \frac{pos}{1} \times \pi_{iter:inner,item}\left(\varrho_{inner:\langle iter,pos\rangle} q_1\right) \\ \mathsf{loop}_v \equiv \pi_{iter} q_v & \mathsf{map} \equiv \pi_{outer:iter,inner}\left(\varrho_{inner:\langle iter,pos\rangle} q_1\right) \\ \multicolumn{2}{l}{\{\ldots, \$v_i \mapsto \pi_{iter:inner,pos,item}\left(q_{v_i} \bowtie_{iter=outer} \mathsf{map}\right), \ldots\} + \{\$v \mapsto q_v\}; \mathsf{loop}_v; \Delta_1 \vdash e_2 \mapsto (q_2, \Delta_2)}\end{array}}{\begin{array}{c}\{\ldots, \$v_i \mapsto q_{v_i}, \ldots\}; \mathsf{loop}; \Delta \vdash \texttt{for } \$v \texttt{ in } e_1 \texttt{ return } e_2 \mapsto \\ \left(\pi_{iter:outer,pos:pos_1,item}\left(\varrho_{pos_1:\langle iter,pos\rangle / outer}\left(q_2 \bowtie_{iter=inner} \mathsf{map}\right)\right), \Delta_2\right)\end{array}}$$

(FOR)

Purely Relational FLWORs

Torsten Grust
Technical University of Munich
Department of Computer Science, Database Systems
Munich, Germany
grust@in.tum.de

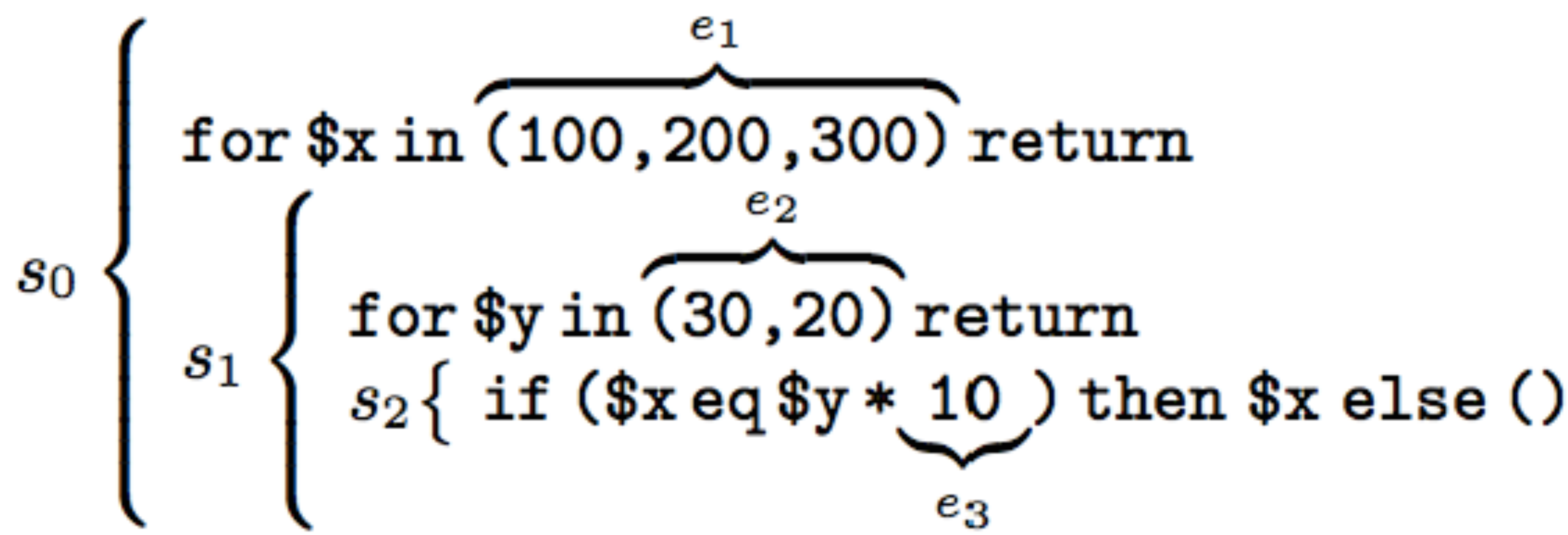
ABSTRACT

We report on a compilation procedure that derives relational algebra plans from arbitrarily nested XQuery FLWOR blocks. While recent research was able to develop relational encodings of trees which may turn RDBMSs into highly efficient XPath and XML Schema processors, here we describe *relational encodings of nested iteration*, *variables*, and the *item sequences* to which variables are bound. The developed techniques are *purely relational* in more than one sense: (a) we rely on a standard (or rather: classical) algebra that is readily supported by relational engines, and (b) we use re-

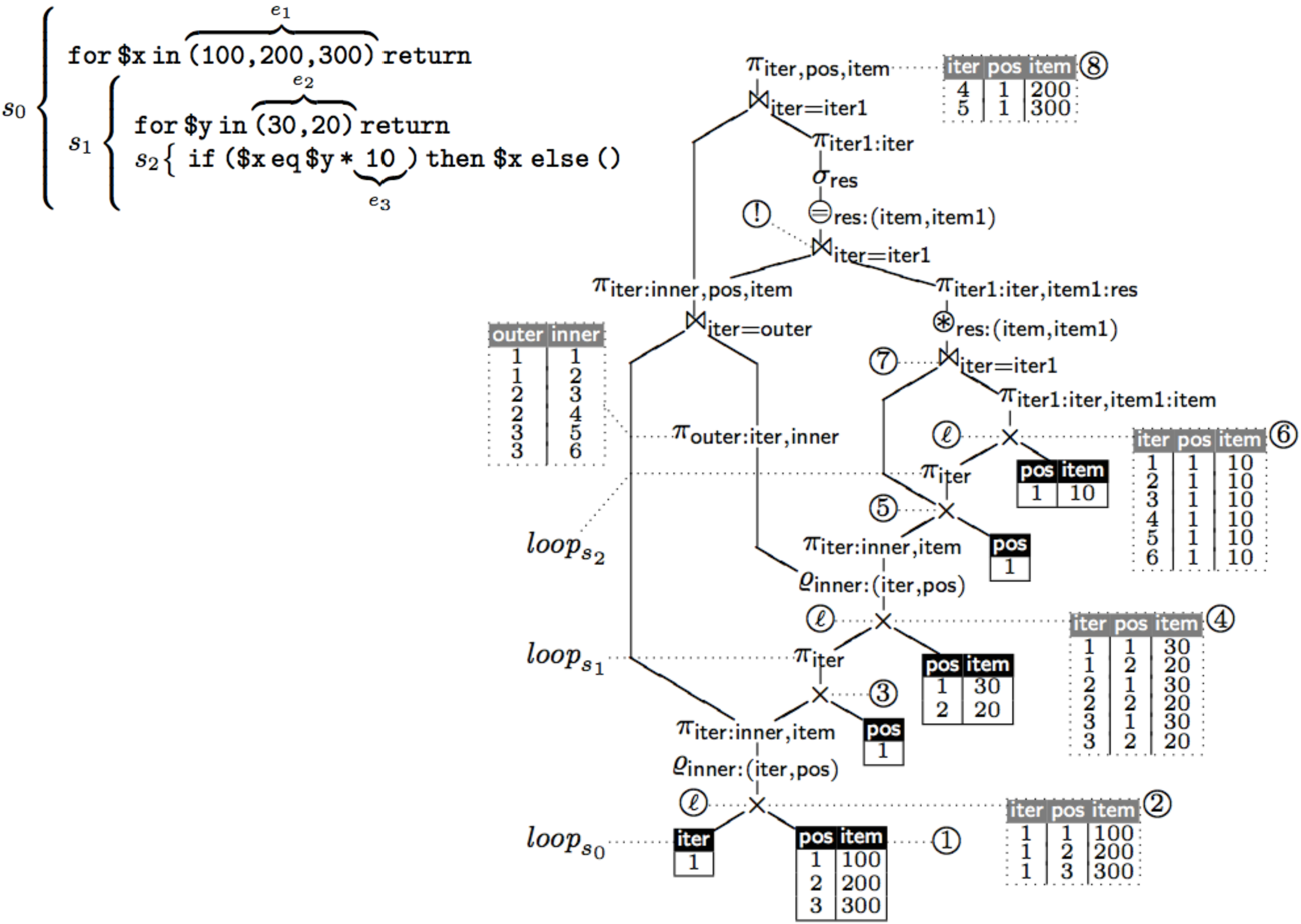
other benefits, the resulting systems inherit the scalability of the underlying relational back-ends [3]. It is legitimate to hope that this technology may be developed into full-fledged XQuery implementations, given that we can find relational ways to also express XQuery concepts beyond XPath axis traversals.

To this end, this paper does *not* talk about XPath evaluation at all but shifts focus to *the* central XQuery language feature, the `for-let-where-order by-return` (or FLWOR) block [2]. The presence of arbitrarily nested iteration as well as the possibility to bind and then refer to variables in

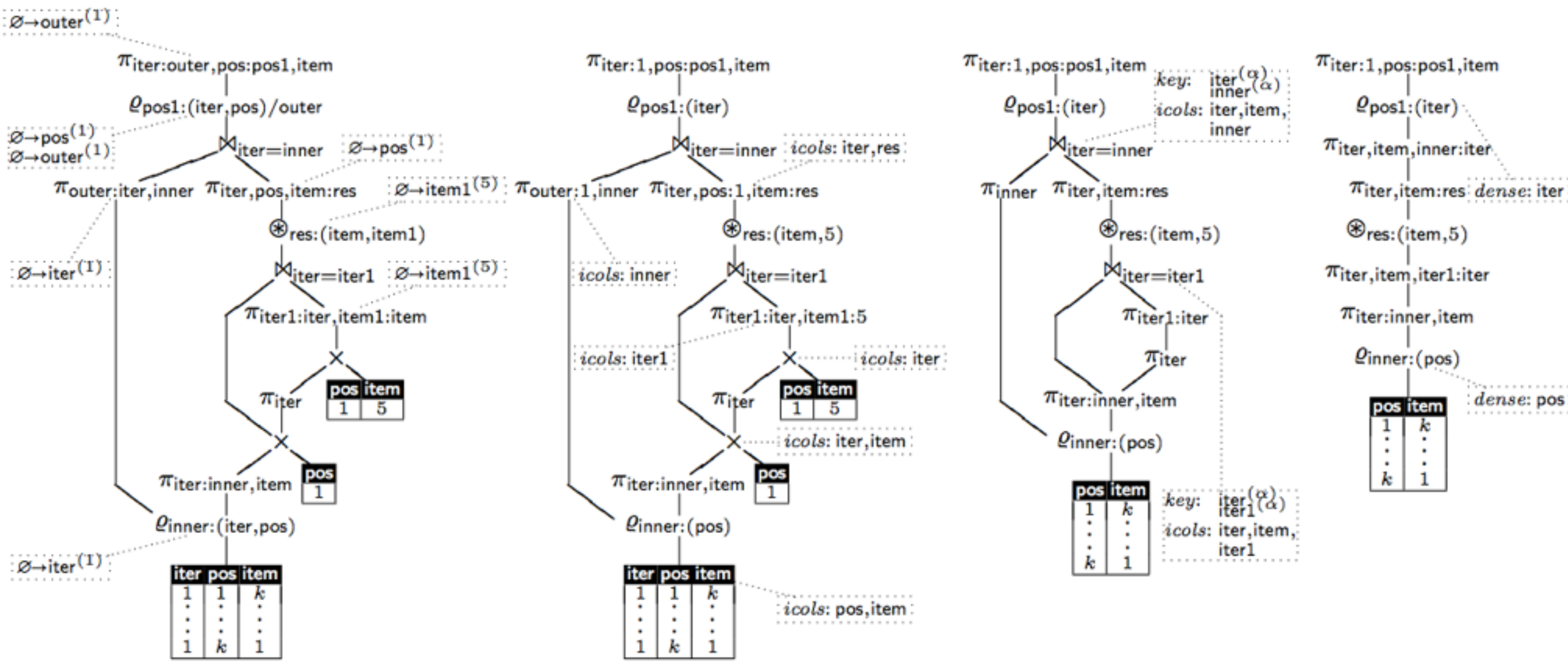
XIME-P 2005, June 16–17, Baltimore, Maryland.



s_0
e_1
for $x in (100,200,300) return
s_1
e_2
for $y in (30,20) return
s_2
if ($x eq $y * 10) then $x else ()
e_3



for $x in (100,200,300) return
for $y in (30,20) return
if ($x eq $y * 10) then $x else ()
e1
e2
e3
s0
s1
s2
π iter,pos,item
⋈ iter=iter1
π iter1:iter
σ res
= res:(item,item1)
!
⋈ iter=iter1
π iter:inner,pos,item
π iter1:iter,item1:res
⋈ iter=outer
* res:(item,item1)
⋈ iter=iter1
π iter1:iter,item1:item
π outer:iter,inner
π iter
ϱ inner:(iter,pos)
π iter:inner,item
π iter
π iter:inner,item
ϱ inner:(iter,pos)
loop s2
loop s1
loop s0
ℓ
1 2 3 4 5 6 7 8
iter pos item: 4 1 200; 5 1 300
outer inner: 1 1; 1 2; 2 3; 2 4; 3 5; 3 6
iter pos item: 1 1 10; 2 1 10; 3 1 10; 4 1 10; 5 1 10; 6 1 10
pos item: 1 10
pos: 1
iter pos item: 1 1 30; 1 2 20; 2 1 30; 2 2 20; 3 1 30; 3 2 20
pos item: 1 30; 2 20
pos: 1
iter: 1
pos item: 1 100; 2 200; 3 300
iter pos item: 1 1 100; 1 2 200; 1 3 300


∅→outer(1)
πiter:outer,pos:pos1,item
ϱpos1:(iter,pos)/outer
∅→pos(1)
∅→outer(1)
⋈iter=inner
∅→pos(1)
πouter:iter,inner
πiter,pos,item:res
∅→item1(5)
⊛res:(item,item1)
∅→iter(1)
⋈iter=iter1
∅→item1(5)
πiter1:iter,item1:item
πiter
pos item
1 5
pos
1
πiter:inner,item
ϱinner:(iter,pos)
∅→iter(1)
iter pos item
πiter:1,pos:pos1,item
ϱpos1:(iter)
⋈iter=inner
icols: iter,res
πouter:1,inner
πiter,pos:1,item:res
⊛res:(item,5)
icols: inner
⋈iter=iter1
πiter1:iter,item1:5
icols: iter1
icols: iter
icols: iter,item
ϱinner:(pos)
icols: pos,item
key: iter(α) inner(α)
icols: iter,item, inner
πinner
πiter,item:res
πiter1:iter
key: iter(α) iter1(α)
icols: iter,item, iter1
πiter,item,inner:iter
dense: iter
πiter,item,iter1:iter
dense: pos

9. REFERENCES

[1] S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Nov. 2003.

[2] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 8(2), 1999.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. Madison, Wisconsin, USA, June 2002.

[4] S. Chien, Z. Vagena, D. Zhang, V. Tsotras, and C. Zaniolo. Efficient Structural Joins on Indexed XML Documents. In *Proc. of the 28th Int'l Conference on Very Large Databases (VLDB)*, Hong Kong, China, Aug. 2002.

[5] D. DeHaan, D. Toman, M. Consens, and M. Öszu. A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding. In *Proc. of the 22nd Int'l ACM SIGMOD Conference on Management of Data*, San Diego, California, USA, June 2003.

[6] M. Fernández, J. Simeon, and P. Wadler. A Semi-monad for Semi-structured Data. In *Proc. of the 8th Int'l Conference on Database Theory (ICDT)*, London, UK, Jan. 2001.

[7] D. Gluche, T. Grust, C. Mainberger, and M. Scholl. Incremental Updates for Materialized OQL Views. In *Proc. of the 5th Int'l Conference on Deductive and Object-Oriented Databases (DOOD)*, Montreux, Switzerland, Dec. 1997.

[8] T. Grust. Accelerating XPath Location Steps. In *Proc. of the 21st Int'l ACM SIGMOD Conference on Management of Data*, Madison, Wisconsin, USA, June 2002.

[9] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *Proc. of the 30th Int'l Conference on Very Large Data Bases (VLDB)*, Toronto, Canada, Aug. 2004.

[10] T. Grust, M. van Keulen, and J. Teubner. Staircase Join: Teach a Relational DBMS to Watch its Axis Steps. In *Proc. of the 29th Int'l Conference on Very Large Databases (VLDB)*, Berlin, Germany, Sept. 2003.

[11] J. Hidders and P. Michiels. Avoiding Unnecessary Ordering Operations in XPath. In *Proc. of the 9th Int'l Workshop on Database Programming Languages (DBPL)*, Potsdam, Germany, Sept. 2003.

[12] R. Krishnamurthy, R. Kaushik, and J. Naughton. XML-to-SQL Query Translation Literature: The State of the Art and Open Problems. In *Proc. of the 1st Int'l XML Database Symposium (XSym)*, Berlin, Germany, Sept. 2003.

[13] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proc. of the 27th Int'l Conference on Very Large Databases (VLDB)*, Rome, Italy, Sept. 2001.

[14] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries over Heterogeneous Data Sources. In *Proc. of the 27th Int'l Conference on Very Large Databases (VLDB)*, Rome, Italy, Sept. 2001.

[15] J. Melton. *Advanced SQL:1999: Understanding Object-Relational and Other Advanced Features.* Morgan Kaufmann Publishers, Amsterdam, 2003.

[16] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. of the 28th Int'l Conference on Very Large Databases (VLDB)*, Hong Kong, China, Aug. 2002.

[17] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. Querying XML Views of Relational Data. In *Proc. of the 27th Int'l Conference on Very Large Databases (VLDB)*, Rome, Italy, Sept. 2001.

7. REFERENCES

[1] C. Beeri, R. Fagin, and J.H. Howard. A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations. In *Proc. of the Int'l ACM SIGMOD Conference on Management of Data*, Toronto, Canada, August 1977.

[2] S. Boag, D. Chamberlin, M.F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, February 2005. W3C Working Draft `http://www.w3.org/TR/xquery/`.

[3] P. Boncz, T. Grust, S. Manegold, J. Rittinger, and J. Teubner. Pathfinder: Relational XQuery Over Multi-Gigabyte Inputs in Interactive Time. Technical Report INS-E0503, CWI, Amsterdam, March 2005.

[4] P. Boncz and M.L. Kersten. MIL Primitives for Querying a Fragmented World. *VLDB Journal*, 8(2), March 1999.

[5] R. Fagin. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transactions on Database Systems (TODS)*, 2(3), September 1977.

[6] T. Grust and S. Klinger. Validation and Type Annotation for Encoded Trees. In *Proc. of the 1st Int'l Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)*, Paris, France, June 2004.

[7] T. Grust, S. Sakr, and J. Teubner. XQuery on SQL Hosts. In *Proc. of the 30th Int'l Conference on Very Large Databases (VLDB)*, Toronto, Canada, September 2004.

[8] T. Grust and J. Teubner. Relational Algebra: Mother Tongue—XQuery: Fluent. In *Proc. of the Twente Data Management Workshop on XML Databases and Information Retrieval (TDM)*, The Netherlands, June 2004.

[9] T. Grust, M. van Keulen, and J. Teubner. Staircase Join: Teach a Relational DBMS to Watch Its Axis Steps. In *Proc. of the 29th Int'l Conference on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.

[10] A. Klug. Calculating Constraints on Relational Expressions. *ACM Transactions on Database Systems (TODS)*, 5(3), September 1980.

[11] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *Proc. of the Int'l ACM SIGMOD Conference on Management of Data*, Paris, France, June 2004.

[12] A. Schmidt, F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. of the 28th Int'l Conference on Very Large Databases (VLDB)*, Hong Kong, China, August 2002.

FERRY — Database–Supported Program Execution

Torsten Grust, Manuel Mayr, Jan Rittinger, and Tom Schreiber

WSI, Universität Tübingen
Tübingen, Germany

⟨*firstname.lastname*⟩@uni-tuebingen.de

```
let e = table Employees (id int, name string,
                         dept string, salary int)
        with keys ((id))
 in for x in e
    group by x.dept
    return (the (x.dept),
             take (2, for y in zip (x.name, x.salary)
                      order by y.2 descending
                      return y))
```


SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
ACM 978-1-60558-551-2/09/06.

Haskell Boards the Ferry

Database-Supported Program Execution for Haskell

George Giorgidze, Torsten Grust, Tom Schreiber, and Jeroen Weijers

Wilhelm-Schickard-Institut für Informatik,
Eberhard Karls Universität Tübingen
{george.giorgidze,torsten.grust,tom.schreiber,jeroen.weijers}@uni-tuebingen.de

2010

[j8] Torsten Grust: **Der Lehrstuhl für Datenbanksysteme am Wilhelm-Schickard-Institut der Universität Tübingen.** Datenbank-Spektrum 10(2): 105-106 (2010)

[j7] Torsten Grust, Jan Rittinger, Tom Schreiber: **Avalanche-Safe LINQ Compilation.** PVLDB 3(1): 162-172 (2010)

[j6] Tom Schreiber, Simone Bonetti, Torsten Grust, Manuel Mayr, Jan Rittinger: **Thirteen New Players in the Team: A Ferry-based LINQ to SQL Provider.** PVLDB 3(2): 1549-1552 (2010)

[c30] Torsten Grust, Manuel Mayr, Jan Rittinger: **Let SQL drive the XQuery workhorse (XQuery join graph isolation).** EDBT 2010: 147-158

[c29] George Giorgidze, Torsten Grust, Tom Schreiber, Jeroen Weijers: **Haskell Boards the Ferry - Database-Supported Program Execution for Haskell.** IFL 2010: 1-18

A study of the exact relationship between DPH and DSH still lies ahead. We conjecture that DSH's loop-lifting compilation strategy does have an equivalent formulation in terms of vectorisation or Blelloch's flattening transformation [4].

$$
\begin{array}{ll}
\mathit{fst}\hat{} & :: [:(a,b):] \rightarrow [:a:] \\
\mathit{snd}\hat{} & :: [:(a,b):] \rightarrow [:b:] \\
\mathit{bpermuteP} & :: [:a:] \rightarrow [:\mathit{Int}:] \rightarrow [:a:]
\end{array}
$$

$$
\begin{array}{l}
\textbf{let}\ sv = [:(1,0.1),(3,1.0),(4,0.0):] \\
\quad\ \ v\ = [:10,20,30,40,50:] \\
\textbf{in} \\
\quad \mathit{sumP}\ (\mathit{snd}\hat{}\ sv *\hat{}\ \mathit{bpermuteP}\ v\ (\mathit{fst}\hat{}\ sv))
\end{array}
$$

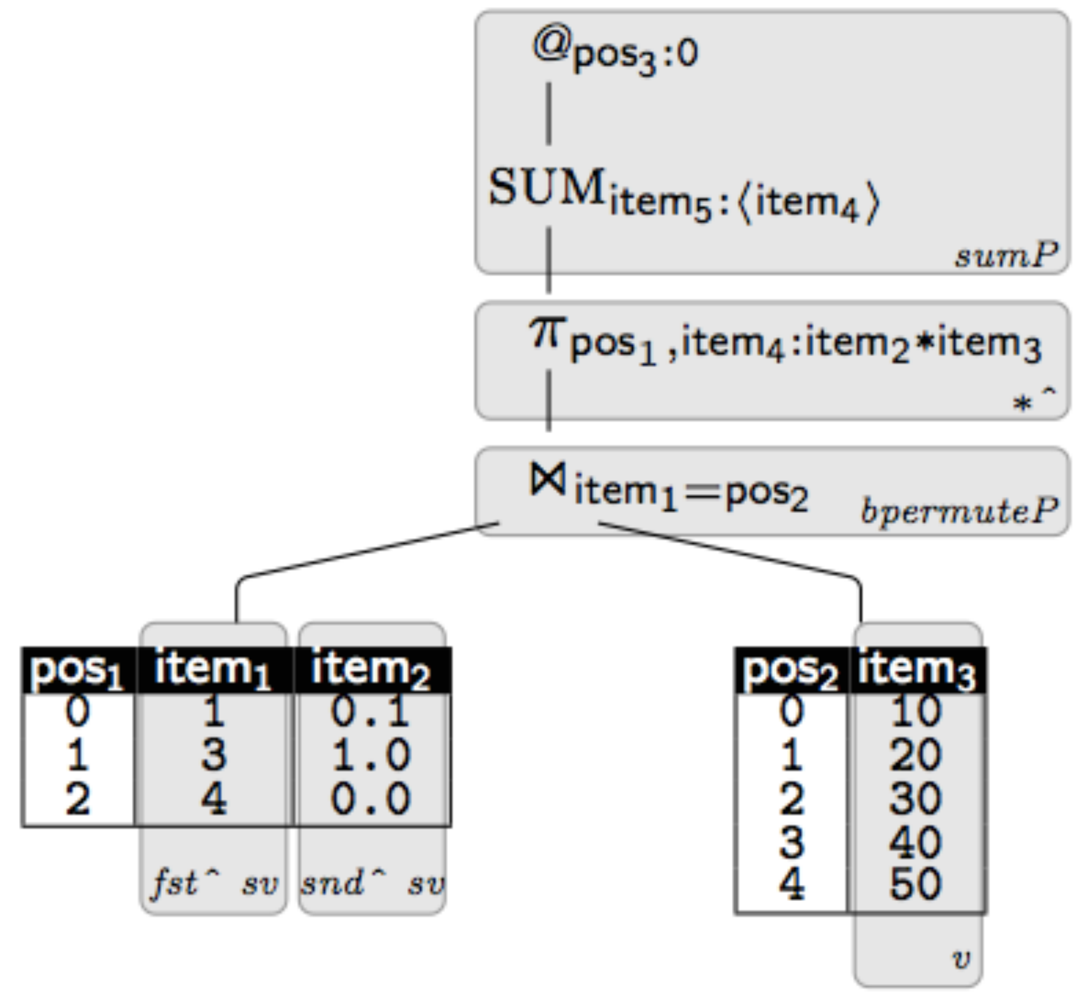


Fig. 6. Intermediate code generated for the sparse vector multiplication example of Fig. 5: DPH (left) *vs.* DSH (right).