XQuery

- Query language for XML data
- includes XPath functionality (path steps)
- iterators, bindings, functions, ...
  (for, if then else, let, sequence, count, doc, ...)
- typing (cast, typeswitch, instance-of, ...)
- xml data construction (element, attribute, ...)

- Example:
  How many sold items cost more than 40?

```
count (for $i in doc("auction.xml")/site/
                 closed_auctions/closed_auction
where $i/price/text() >= 40
return $i/price)
```

Evaluating XQuery expressions with MonetDB

Jan Rittinger
CWI Amsterdam & University of Konstanz

Summerschool GK University of Konstanz
St. Vigil (Italy)

Motivation

- about me
  - studying in Konstanz
  - Bachelor thesis in database and information systems department (pathfinder project)
  - currently working at the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam (the Netherlands)
  - database and multimedia retrieval department

- project: MonetDB
- plan for pathfinder: using a database for the query execution
- my task at the CWI: building a first XQuery prototype based on pathfinder using MonetDB

MonetDB (http://monetdb.cwi.nl/)

- Main-Memory database
- main focus on cache conscious algorithms
- uses only binary tables
- open source software
- low level interpreter language (MIL)
  - good extensibility
    (simple interface for c-functions)
  - database operators
  - type system
  - iterative operators (e.g batloop, while, ...)
  - user-defined functions

idea

- take pathfinder and start from the typed XQuery Core representation

  ➔ core tree

  ... relational operators

- solution: XQuery for SQL-Hosts

  (Grust, Sakr, Teubner - VLDB 2004)

- represent every intermediate result by a table with the schema iter|pos|item

the iter|pos|item schema

10

pos	item
1	10

(10, 20, 30)

iter	pos	item
1	1	10
1	2	20
1	3	30

- iter is needed to support the looping primitive **for**

  ➔ that is why everything with this schema is called **loop-lifted**

understanding the example

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- the sequence (10,20) is evaluated ...
- ... and bound to variable $a

- the sequence (30,40) is evaluated ...
- ... and each value of the sequence is iteratively bound to variable $b

- for each binding the return sequence with the actual variable bindings is evaluated

understanding the example

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

iter	pos	item
1	1	10
1	2	20
1	3	30
1	4	10
1	5	20
1	6	40

translation of constants

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- for each constant a new table is created
- the table is expanded to the current scope (cross-product with $loop_0$)

iter
1

$loop_0$

$loop_0$ x

pos	item
1	10

=

iter	pos	item
1	1	10

$loop_0$ x

pos	item
1	20

=

iter	pos	item
1	1	20

translation of sequences

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- a (special) union is made to combine the two inputs
- the pos column gets new dense numbers within each iter

iter	pos	item
1	1	10

iter	pos	item
1	1	20

iter	pos	item
1	1	10
1	2	20

translation of let expressions

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- the variable is added to the environment Γ with the value of its expression

$a :=

iter	pos	item
1	1	10
1	2	20

➔

iter	pos	item
1	1	10
1	2	20

$a

translation of iteration body

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

iter
1

$loop_0$

$loop_0$ x

pos	item
1	30

=

iter	pos	item
1	1	30

$loop_0$ x

pos	item
1	40

=

iter	pos	item
1	1	40

↓

iter	pos	item
1	1	30
1	2	40

translation of for expression (1/3)

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- $b is successively bound to the values (30, 40) - The return expression ($a, $b) is evaluated for each binding
- XQuery is a functional-style language

  ➔ It is sound to evaluate ($a, $b) for all bindings in parallel

$b :=

iter	pos	item
1	1	30
2	1	40

➔

iter	pos	item
1	1	30
2	1	40

$b

translation of for expression (2/3)

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- every variable in the environment ($a) needs to be mapped to the new scope

iter	pos	item
1	1	30
2	1	40

$b

➔

iter
1
2

$loop_1$

➔

outer	inner
1	1
1	2

map_1

translation of for expression (3/3)

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

- every variable in the environment ($a) needs to be mapped to the new scope

iter	pos	item
1	1	10
1	2	20

$a

|x|

outer	inner
1	1
1	2

map_1

=

iter	pos	item
1	1	10
1	2	20
2	1	10
2	2	20

$a

translation of the for body

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

iter	pos	item
1	1	10
1	2	20
2	1	10
2	2	20

$a

iter	pos	item
1	1	30
2	1	40

$b

iter	pos	item
1	1	10
1	2	20
1	3	30
2	1	10
2	2	20
2	3	40

backmapping of the result

```
let $a := (10,20) return
for $b in (30,40) return ($a,$b)
```

iter	pos	item
1	1	10
1	2	20
1	3	30
2	1	10
2	2	20
2	3	40

|x|

inner	outer
1	1
2	1

$\mathtt{map}_1$

=

iter	pos	item
1	1	10
1	2	20
1	3	30
1	4	10
1	5	20
1	6	40

conclusion

- almost every feature of XQuery is supported
- 18 of 20 Xmark queries are running with generated MIL code
- developed some new ideas, which work nicely
- discovered some problems/bugs in the pathfinder code

but ...

- performance problems (mostly because of iterative translations)

  ➔ solution: intelligent loop-lifted version
- a lot of work has still to be done (e.g. built-in functions, ...)