Sequence comprehension for XQuery's FLWOR expression

Maarten Fokkinga

Version of May 9, 2003, 15:14

Abstract. In XQuery the FLWOR expression is a kind of comprehension for sequences. It differs from sequence comprehensions as found in functional programming languages mainly in the presence of an *order by* clause. We define a sequence expression with an *order by* clause that is suitable to express the semantics of XQuery's FLWOR expressions.

1 Introduction. In XQuery the FLWOR expression constructs a sequence out of given sequences. It resembles the set comprehension '$\{\ldots \mid \ldots\}$' and the related sequence comprehension in functional programming languages. There is, however, one striking difference: the *order by* clause. This clause uses the bound variable, and has an overall effect on the order of the resulting sequence. We define a sequence expression with an *order by* clause that is suitable to express the semantics of XQuery's FLWOR expressions. We do so with particular effort to define the semantics with well-known, general purpose operations; in particular, a sequence re-arranging operation $\lessgtr$. Wherever possible we use the Z notation to express mathematics. (So, a sequence xs over a set S is a function from $1 \mathrel{..} \#xs$ to S where the ith item in the sequence is given by the function application $xs\,i$. Hence, $\operatorname{dom} xs$ is the set $1 \mathrel{..} \#xs$ and $\operatorname{ran} xs$ is the set of items in xs. Brackets $\langle$ and $\rangle$ are used to display sequences.)

2 The FLWOR expression. The basic form of the FLWOR expression is (discarding the 'L' part for the *let* clause):

`for $x in` xs
`where` P
`order by` E
`return` F

Here, xs is a sequence, P is a predicate expression, E is an expression evaluating to a value in an ordered set, and F is a value expression. All of P, E, and F may contain occurrences of `$x`. The entire expression denotes a sequence that is obtained as follows:

Consider sequence xs.
Let x (read: `$x`) range over the items in xs, in the order given by xs.
Discard the x's for which P evaluates to *false*.
Re-arrange the resulting sequence in such a way that later items have larger E-values, keeping the original order where the E-values are equal.
For each value x in the sequence thus obtained, put the F-value in the result sequence.

We shall define a sequence comprehension '$\langle x :: xs \mid P \lessgtr E \bullet F \rangle$' that exactly equals the meaning of the above FLWOR expression. Here, symbols $::$, $|$, $\lessgtr$, $\bullet$ are syntactic separators.

3 Ordering. First some auxiliary *order* concepts that are interesting by themselves.

- A pre-order $\preceq_f$ constructed out of a (pre-)order $\preceq$:

$$x \preceq_f y \quad \Leftrightarrow \quad f\,x \;\preceq\; f\,y$$

  Equivalently, $\preceq_f$ equals $f \mathbin{;} (\preceq) \mathbin{;} f^{\sim}$. Pre-order $\preceq_f$ is an order if $\preceq$ is so *and* f is injective. Order $\preceq_f$ is total if $\preceq$ is so *and* f is injective. Note also that $(\preceq_f)_g$ equals $\preceq_{f \circ g}$.

- The combination of two pre-orders with priority for the first one. Let $\preceq_1$ and $\preceq_2$ be pre-orders over X. Then $(\preceq_1) \oplus (\preceq_2)$ is the pre-order $\preceq$ defined by:

$$x \preceq y \quad \Leftrightarrow \qquad x \preceq_1 y \wedge x \not\succeq_1 y \quad \vee \quad ((x \preceq_1 y \Leftrightarrow x \succeq_1 y) \wedge x \preceq_2 y)$$

  In other words, x is at most y in the composite order if x is smaller than y in the first order, and only if x and y are unrelated or equal in the first order, then the second order comes into play. Note that if the first order is total, then the second doesn't matter; and if the second order is total then so is the composite order. Note also that $(\preceq_{fst}) \oplus (\preceq_{snd})$ is the lexicographic order on tuples. A variation is to use two orders: $(\preceq_{fst}) \oplus (\leq_{snd})$; this one turns up in the sequel.

- The predicate that a sequence is totally (pre-)ordered with respect to a pre-order $\preceq$:

$$\begin{aligned} (\preceq)\ \underline{orders}\ xs \quad &\Leftrightarrow \quad (\leq) \mathbin{;} xs \subseteq xs \mathbin{;} (\preceq) \\ &\Leftrightarrow \quad \forall\, i, j : \mathrm{dom}\, xs \mid i \leq j \bullet xs\, i \preceq xs\, j \end{aligned}$$

  That is, "going in xs to a larger index yields a larger item". We might also say that "xs is ascending with respect to $\preceq$".

Using these concepts we define the *ordering* operation $\lessgtr$ that re-arranges a sequence. Let $\preceq$ be a pre-order on $\mathrm{ran}\, xs$. Then $xs \lessgtr (\preceq)$ is an ordered re-arrangement of xs such that $\preceq$ is obeyed where applicable, and the original order is preserved otherwise:

$xs \lessgtr (\preceq) \quad = \quad$ Let ys be equal to xs except that each item is tupled with its index:
$\qquad ys = \{i : \mathrm{dom}\, xs \mid i \mapsto (xs\, i,\ i)\}$.
Let zs be the uniquely determined sequence such that:
$\qquad zs$ is a permutation of ys, and $((\preceq_{fst}) \oplus (\leq_{snd}))$ $\underline{orders}$ zs.
Take as result: $fst \circ zs$ (discarding the added indexes).

4 Sequence comprehension with an *order by* clause. Let xs be a sequence. Let P be a predicate, E and F be expressions; these may contain free occurrences of x. Then '$\langle x :: xs \mid P \lessgtr E \bullet F\rangle$' is our syntax for sequence comprehension. To define the semantics, assume without loss of generality that i doesn't occur free in P, E, or F. Then:

$$\begin{aligned} \langle x :: xs \mid P \lessgtr E \bullet F\rangle \quad = \quad & xs \ \upharpoonright p \ \lessgtr (\leq_e) \ \mathbin{;} f \\ \text{where } p \ &= \ (\lambda\, x : \mathrm{ran}\, xs \bullet P) \\ e \ &= \ (\lambda\, x : \mathrm{ran}\, xs \bullet E) \\ f \ &= \ (\lambda\, x : \mathrm{ran}\, xs \bullet F) \end{aligned}$$

Here we have used the Z notation for filtering of a sequence xs by a predicate p, namely: $xs \upharpoonright p$. Moreover, $xs \mathbin{;} f = f \circ xs = (\lambda\, i : \mathrm{dom}\, xs \bullet f(xs\, i))$ = "map" f over xs.

We leave it as an exercise for the reader to define the/a semantics of '$\langle x :: xs;\ y :: ys \mid \ldots\rangle$'.

* * *