XQuery Syntax in HXQ

Leonidas Fegaras
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76019
fegaras@cse.uta.edu
http://lambda.uta.edu/HXQ/

January 4, 2009

Symbols in **blue font** are lexical tokens (terminals), symbols in regular font are either meta-symbols or non-terminals. Here is the meaning of the meta-symbols (ϵ matches the empty input):

(a)	=	a	
a b	=	a then b	(concatenation)
a | b	=	either a or b	(alternation)
[a]	=	a | ϵ	(optionality)
{ a }	=	a | a a | a a a | …	(repetition)
{ a ,}	=	a | a **,** a | a **,** a **,** a | …	
{ a ;}	=	a | a **;** a | a **;** a **;** a | …	
{, a }	=	ϵ | a | a **,** a | a **,** a **,** a | …	

```
query       ::=  { declare variable var := e                                (a variable declaration)
                  | declare function qname ( { var [ as type ] ,} )
                       [ as type ] { e }                                     (a function declaration)
                  | e ;}                                                     (an XQuery)
qname       ::=  [ id : ] id                                                 (a qualified name is namespace:localname)
var         ::=  $ id                                                        (variables should begin with $)
string      ::=  " { { { e ,} } | char } "                                   (you may embed XQuery values in a string)
type        ::=  qname [ () ] [ * | + ]                                      (types are currently ignored)
e           ::=  ( for fbinds | let lbinds ) { for fbinds | let lbinds }
                       [ where e ] [ orderby ] return e                      (FLOWR expression)
                 | some fbinds satisfies e                                   (existential quantification)
                 | every fbinds satisfies e                                  (universal quantification)
                 | if e then e else e
                 | insert e into e                                           (insert the former inside the latter)
                 | delete from e                                             (remove from parent)
                 | replace e with e                                          (replace the former with the latter)
                 | @ step predicates
                 | step predicates { path }                                  (an XPath path)
                 | element                                                   (element construction)
                 | e binop e                                                 (binary operation)
                 | unop e                                                    (unary operation)
                 | e instance of type                                        (type check (ignored))
                 | integer                                                   (integer constant)
                 | double                                                    (floating point)
                 | string
fbinds      ::=  { var [ at var ] in e ,}                                    (for-bindings)
lbinds      ::=  { var := e ,}                                               (let-bindings)
orderby     ::=  order by { e [ ascending | descending ] ,}                  (default is ascending)
binop       ::=  to | + | - | * | div | idiv | mod | = | != | < | <=
                 | > | >= | << | >> | is | eq | ne | lt | le | gt | ge
                 | and | or | not | union | intersect | except
unop        ::=  + | - | not
element     ::=  < qname { qname = string } > content </ qname >
                 | < qname { qname = string } />                             (empty element)
                 | element ( qname | { e } ) { { e ,} }
                 | attribute ( qname | { e } ) { { e ,} }
content     ::=  { { { e ,} } | string | text | element }
path        ::=  / step predicates                                           (child-of)
                 | // step predicates                                        (descendant-of)
                 | /@ step predicates                                        (attribute-of)
                 | //@ step predicates                                       (descendant-attribute-of)
                 | /.. predicates                                            (parent-of)
predicates  ::=  { [ e ] }
step        ::=  var
                 | qname [ :: ( qname | * ) ]                                (an XPath step is axis::test)
                 | .                                                         (current context)
                 | *                                                         (any name)
                 | ( {, e } )                                                (sequence construction)
                 | qname ( {, e } )                                          (function call)
```

Figure 1: XQuery BNF