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 ($\epsilon$ matches the empty input):

- $$(a) = a$$
- $$(a \ b) = a \text{ then } b$$ (concatenation)
- $$a \mid b = \text{ either } a \text{ or } b$$ (alternation)
- $$[a] = a \mid \epsilon$$ (optionality)
- $$\{a\} = a \mid a \mid a \mid a \mid \ldots$$ (repetition)
- $$\{a,\} = a \mid a, a \mid a, a \mid \ldots$$
- $$\{a:\} = a \mid a ; a \mid a ; a \mid \ldots$$
- $$\{, a\} = \epsilon \mid a \mid a, a \mid a, a \mid a \mid \ldots$$
query ::= { declare variable var [ as type ] := e
declare function qname ( { var [ as type ] }, ) [ as type ] { e }
declare view qname ( { , var } ) { e }
ed ;
}(a variable declaration)

qname ::= [ id : ] id
(a qualified name is namespace:localname)

var ::= $ id
(variables should begin with $)

type ::= qname [ ( [ qname | * ] [ , qname ] ) | * | + | ? ]
(XQuery type)

e ::= ( for fbinds | let lbinds ) { for fbinds | let lbinds }
[ where e ] [ orderby ] return e
some fbinds satisfies e
every fbinds satisfies e
if e then e else e
insert e into e
delete from e
replace e with e
@ step predicates
step predicates { path }
element
e binop e
unop e
e instance of type
e cast as type
e castable as type
typeswitch ( e ) typecases
integer
double
string
(FLOWR expression)
(existential quantification)
(universal quantification)
(insert the former inside the latter)
(remove from parent)
(replace the former with the latter)
(an XPath path)
(element construction)
(binary operation)
(unary operation)
(type check)
(type cast)
(can be cast to type?)
(type switch)
(integer constant)
(floating point)

fbinds ::= { var [ at var ] in e ,}
(for-bindings)

lbinds ::= { var := e ,}
(let-bindings)

orderby ::= order by { e [ ascending | descending ] ,}
(default is ascending)

typecases ::= { case type return e } default return e
(type cases)

binop ::= to | + | - | * | div | idiv | mod | = | != | < | <= |
> | >= | <= | == | eq | ne | lt | le | gt | ge |
and | or | not | union | intersect | except

unop ::= + | - | not

element ::= < qname { qname = string } > content <*> qname >
< qname { qname = string } />

(element)

content ::= { { e , } | string | text | element }

path ::= / step predicates

(path)

step ::= var

(step)

Figure 1: XQuery BNF