Assertion grammar

January 19, 2021

1 Grammar with infix operators

```
<assertion contract> ::= <module> +
<module> ::= entrypoint ['%' <IDENT>] <pattern> '=' <assertion>
<pattern> ::=
  | '_'
  | nil
  | none
  | <var declaration>
  | '(' <prim pattern> ')'
<var declaration>::= '(' IDENT ':' <type> ')'
<prim pattern> ::=
  | pair <pattern> <pattern>
  | left <pattern>
  | right <pattern>
  | cons <pattern> <pattern>
  | some <pattern>
<type> ::=
  | int
  | bool
  | bytes
  | string
  | mutez
  | nat
  | address
  | chain_id
  | key
  | key_hash
  | operation
  | signature
  | timestamp
  | unit
  | list <type>
  | set <type>
  | option <type>
  | or <type> <type>
  | pair <type> <type>
```

```
   | lambda <type> <type>
   | map <type> <type>
   | contract <type>
   | big_map <type> <type>
   | '(' <type> ')'
<assertion> ::=
   | <quantifier> <var declaration> <assertion>
   | if <expression> then <assertion>
   | assert <expression>
<quantifier> ::= forall | exists
<atom> ::=
   | IDENT
   | <constant>
   | '(' <expression> ')'
<expression> ::=
   | <atom>
   | nth <expression> <expression>
   | <expression> <binop> <expression>
   | <unop> <atom>
   | slice <expression> <expression> <expression>
   | if <expression> then <expression> else <expression>
<constant> ::=
   | <bool literal>
   | INT_LITERAL
   | STRING_LITERAL
<bool literal> ::= true | false
<binop> ::=
   | '+'
   | '-'
   | '*'
   | '/'
   | '%'
   | "||"
   | "&&"
   | "xor"
   | "<<"
   | ">>"
   | '='
   | "<>"
   | '<'
   | '>'
   | ">="
   | "<="
<unop> ::= not | size | abs | '-' | '+'
```

Listing 1: Assertion grammar; Infix flavor

1.1 Examples

```
(* Example contract:
 * parameter (or (pair (list int) int %A)
 *               (or (pair (list int) string %B)
 *                   (or (option bool %C) string %D)))
 *)

(* %A: assert if input list is sorted *)
entrypoint %A (pair (p : list int) _ ) =
    forall (n: int)
        forall (m : int)
            if (n < (size p) && m < (size p)) then
                if (n > m) then
                    assert (nth n p) > (nth m p)

(* %B: simple assertion without any quantifiers *)
entrypoint %B (pair (l : list int) _ ) =
    if (size l) > 10 then
        assert (nth 9 l) = 7

(* %C: assertion without an "assert" expression *)
entrypoint %C (some (i : option bool)) =
    assert i

(* %D: assertion with the slice expression *)
entrypoint %D (s : string) =
    if (size s) > 0 then
        assert (slice 0 (size s) s) = s
```

═══════ OR ═══════

```
(* %A: assert if input list is sorted *)
entrypoint (left (pair (p : (list int)) _ )) =
    forall (n : int)
        forall (m : (list int))
            if (n < size p) && ( m < size p) then
                if n > m then
                    assert (nth n p) > (nth m p)

(* %B: simple assertion without any quantifiers *)
entrypoint (right (left (pair (l : (list int) _ )))) =
    if (size l) > 10 then
        assert (nth 9 l) = 7

(* %C: assertion without an "assert" expression *)
entrypoint (right [right (left (some [i : bool]))]) =
    assert i

(* %D: assertion with the slice expression *)
```

```
entrypoint (right [right (right (s : string))]) =
    if (size s) > 0 then
        assert (slice 0 (size s) s) = s
```

Listing 2: Example contracts; Infix flavor

2 Grammar with prefix operators (Michelson flavor)

```
<assertion contract> ::= <module> +
<module> ::=
  | '(' entrypoint ['%' <IDENT>] <pattern> <assertion app> ')'
  | '[' entrypoint ['%' <IDENT>] <pattern> <assertion app> ']'
<pattern> ::=
  | '_'
  | nil
  | none
  | <var declaration>
  | '(' <prim pattern> ')'
  | '[' <prim pattern> ']'
<var declaration> ::=
  | '(' IDENT ':' <type> ')'
  | '[' IDENT ':' <type> ']'
<prim pattern> ::=
  | pair <pattern> <pattern>
  | left <pattern>
  | right <pattern>
  | cons <pattern> <pattern>
  | some <pattern>
<assertion app> ::=
  | '(' <assertion> ')'
  | '[' <assertion> ']'
<assertion> ::=
  | <quantifier> <var declaration> <assertion app>
  | if <expression> <assertion app>
  | assert <expression>
<quantifier> ::= forall | exists
<type> ::=
  | int
  | bool
  | bytes
  | string
  | mutez
  | nat
  | address
  | chain_id
  | key
  | key_hash
  | operation
  | signature
  | timestamp
  | unit
  | '(' <composite type> ')'
  | '[' <composite type> ']'
```

```
<composite type> ::=
  | list <type>
  | set <type>
  | option <type>
  | or <type> <type>
  | pair <type> <type>
  | lambda <type> <type>
  | map <type> <type>
  | contract <type>
  | big_map <type> <type>
<expression> ::=
  | IDENT
  | <constant>
  | '(' if <expression> <expression> <expression> ')'
  | '[' if <expression> <expression> <expression> ']'
  | <primitive app>
<constant> ::=
  | <bool literal>
  | INT_LITERAL
  | STRING_LITERAL
<bool literal> ::= true | false
<prim app> ::=
  | '(' <primitive> ')'
  | '[' <primitive> ']'
<primitive> ::=
  | <unop> <expression>
  | <binop> <expression> <expression>
  | slice <expression> <expression> <expression>
  | <expression>
<unop> ::=
  | size
  | abs
  | neg
  | not
<binop> ::=
  | nth
  | add
  | sub
  | mul
  | div
  | mod
  | or
  | and
  | xor
  | lsl
  | lsr
  | eq
  | neq
  | lt
  | gt
```

```
    | le
    | ge
```

Listing 3: Assertion grammar; Michelson flavor

2.1 Examples

```
(* Example contract:
 * parameter (or (pair (list int) int %A)
 *               (or (pair (list int) string %B)
 *                   (or (option bool %C) string %D)))
 *)

(* %A: assert if input list is sorted *)
(entrypoint %A (pair (p : (list int)) _ )
    (forall (n : int)
        (forall (m : (list int))
            (if (and (lt n (size p)) (lt m (size p)))
                (if (lt n m)
                    (assert (gt (nth n p) (nth m p))))))))

(* %B: simple assertion without any quantifiers *)
(entrypoint %B (pair (l : (list int)) _ ))
    (if (gt (size l) 10)
        (assert (eq (nth 9 l) 7))))

(* %C: assertion without an "assert" expression *)
(entrypoint %C (some [i : bool])
    [assert i])

(* %D: assertion with the slice expression *)
(entrypoint %D (s : string)
    (if (gt (size s) 0)
        (assert (eq (slice 0 (size s) s) s))))
```

═══════ OR ═══════

```
(* %A: assert if input list is sorted *)
(entrypoint (left (pair (p : (list int)) _ ))
    (forall (n : int)
        (forall (m : (list int))
            (if (and (lt n (size p)) (lt m (size p)))
                (if (lt n m)
                    (assert (gt (nth n p) (nth m p))))))))

(* %B: simple assertion without any quantifiers *)
(entrypoint (right (left (pair (l : (list int)) _ ))))
    (if (gt (size l) 10)
        (assert (eq (nth 9 l) 7))))

(* %C: assertion without an "assert" expression *)
(entrypoint (right [right (left (some [i : bool]))])
    [assert i])

(* %D: assertion with the slice expression *)
```

```
(entrypoint (right [right (right (s : string))])
    (if (gt (size s) 0)
        (assert (eq (slice 0 (size s) s) s))))
```

Listing 4: Example contracts; Michelson flavor