```
┌──────────────────────────── MODULE TESpec ────────────────────────────┐
EXTENDS Naturals, TLC, FiniteSets

CONSTANTS CONTRACTS,     set of contracts in Tezos
          TOKENS,        set of token contracts
          EXCHANGE,      exchange contract name
          INIT_TOKEN,    initial token amount
          INIT_XTZ       initial (mu)xtz amount

VARIABLES xtzMap,      XTZ amount state of contracts
          tokenMap,    token amount state of contracts
          orders       orders state

├───────────────────────────────────────────────────────────────────────┤
 some common helper operators

Range(T) ≜ {⟨T[x], x⟩ : x ∈ DOMAIN T}
Pick(S) ≜ CHOOSE s ∈ S : TRUE

RECURSIVE SetReduce(_, _, _)
SetReduce(Op(_, _), S, value) ≜
  IF S = {} THEN value
   ELSE LET s ≜ Pick(S)
          IN   IF Op(s[1], value) = Op(value, s[1])
           THEN SetReduce(Op, S \ {s}, Op(s[1], value))
           ELSE  Assert(FALSE, "error")

Sum(S) ≜ LET _op(a, b) ≜ a + b
           IN    SetReduce(_op, S, 0)

├───────────────────────────────────────────────────────────────────────┤
 some exchange helper operators

Buyers ≜
  {x ∈ CONTRACTS : xtzMap[x] > 0 ∧ x ≠ EXCHANGE}

Sellers(token) ≜
  {x ∈ CONTRACTS : tokenMap[token][x] > 0 ∧ x ≠ EXCHANGE}

PickOrder(key) ≜
  LET matches ≜ {x ∈ orders : x.key = key}
  IN    IF matches = {} THEN [xtz ↦ 0, token ↦ 0]
         ELSE CHOOSE m ∈ matches : TRUE

XTZTransfer(owner, receiver, amount) ≜
  IF owner = receiver
   THEN xtzMap
   ELSE [x ∈ CONTRACTS ↦
```

```
              CASE x = owner → xtzMap[x] − amount
                □   x = receiver → xtzMap[x] + amount
                □   OTHER  → xtzMap[x]]

TOKENTransfer(token, owner, receiver, amount) ≜
  IF owner = receiver
   THEN tokenMap
   ELSE [t ∈ TOKENS ↦
           [x ∈ CONTRACTS ↦
             IF t = token
              THEN CASE x = owner → tokenMap[t][x] − amount
                      □   x = receiver → tokenMap[t][x] + amount
                      □   OTHER  → tokenMap[t][x]
              ELSE  tokenMap[t][x]]]
```

tez.exchange basic user operators

```
CreateBuyingOrder(token, buyer, price, xtz_amount) ≜
  LET key ≜ ⟨buyer, token, TRUE, price⟩
      order ≜ PickOrder(key)
      prev_xtz_amount ≜ order.xtz
  IN
   ∧ xtzMap′ = XTZTransfer(buyer, EXCHANGE, xtz_amount)
   ∧ orders′  = {x ∈ orders : x.key ≠ key} ∪
                  {[key ↦ key, xtz ↦ xtz_amount + prev_xtz_amount]}
   ∧ UNCHANGED ⟨tokenMap⟩


ExecuteBuyingOrder(order, executer, token_amount) ≜
  LET token  ≜ order.key[2]
      price  ≜ order.key[4]
      owner  ≜ order.key[1]
      consumed_xtz ≜ price ∗ token_amount
      remain_xtz ≜ order.xtz − consumed_xtz
  IN
   ∧ remain_xtz ≥ 0
   ∧ xtzMap′ = XTZTransfer(EXCHANGE, executer, consumed_xtz)
   ∧ tokenMap′ = TOKENTransfer(token, executer, owner, token_amount)
   ∧ orders′ = IF remain_xtz = 0
                 THEN {x ∈ orders : x.key ≠ order.key}
                 ELSE {x ∈ orders : x.key ≠ order.key} ∪
                        {[key ↦ order.key, xtz ↦ remain_xtz]}

CreateSellingOrder(token, seller, price, token_amount) ≜
```

```
  $\text{LET } key \triangleq \langle seller, token, \text{FALSE}, price \rangle$
        $order \triangleq PickOrder(key)$
        $prev\_token\_amount \triangleq order.token$
  $\text{IN}$
  $\land tokenMap' = TOKENTransfer(token, seller, EXCHANGE, token\_amount)$
  $\land orders' = \{x \in orders : x.key \neq key\} \cup$
                 $\{[key \mapsto key, token \mapsto token\_amount + prev\_token\_amount]\}$
  $\land \text{UNCHANGED } \langle xtzMap \rangle$

$ExecuteSellingOrder(order, executer, xtz\_amount) \triangleq$
  $\text{LET } token \triangleq order.key[2]$
        $price \triangleq order.key[4]$
        $owner \triangleq order.key[1]$
  $\text{IN}$
  $\land price \neq 0$
  $\land \text{LET } consumed\_token \triangleq xtz\_amount \div price$
          $remain\_token \triangleq order.token - consumed\_token$
     $\text{IN}$
     $\land remain\_token \geq 0$
     $\land xtzMap' = XTZTransfer(executer, owner, xtz\_amount)$
     $\land tokenMap' = TOKENTransfer(token, EXCHANGE, executer, consumed\_token)$
     $\land orders' = \text{IF } remain\_token = 0$
                    $\text{THEN } \{x \in orders : x.key \neq order.key\}$
                    $\text{ELSE } \{x \in orders : x.key \neq order.key\} \cup$
                           $\{[key \mapsto order.key, token \mapsto remain\_token]\}$
```

---

some invariants for checking

```
$xtzMapChecker \triangleq$
  $Sum(Range(xtzMap)) = (Cardinality(CONTRACTS) - 1) * INIT\_XTZ$

$tokenMapChecker \triangleq$
  $[t \in TOKENS \mapsto Sum(Range(tokenMap[t]))] =$
    $[t \in TOKENS \mapsto (Cardinality(CONTRACTS) - 1) * INIT\_TOKEN]$

$ordersChecker \triangleq$
  $\land xtzMap[EXCHANGE] =$
        $Sum(\{\langle order.xtz, order.key \rangle : order \in$
               $\{x \in orders : x.key[3] = \text{TRUE}\}\})$

  $\land [t \in TOKENS \mapsto tokenMap[t][EXCHANGE]] =$
      $[t \in TOKENS \mapsto$
        $Sum(\{\langle order.token, order.key \rangle : order \in$
             $\{x \in orders : x.key[3] = \text{FALSE} \land x.key[2] = t\}\})]$
```

---

the init behavior

```
Init ≜
  ∧ xtzMap = [x ∈ CONTRACTS ↦ IF x = EXCHANGE
                               THEN 0
                               ELSE  INIT_XTZ]
  ∧ tokenMap = [t ∈ TOKENS ↦
                  [x ∈ CONTRACTS ↦ IF x = EXCHANGE
                                    THEN 0
                                    ELSE  INIT_TOKEN]]
  ∧ orders = {}
```

---

the next behavior

this behavior will pick random token and *executer* to test possible operations

```
Next ≜
  LET token ≜ RandomElement(TOKENS)
      Inside(t) ≜
        LET seller ≜ RandomElement(Sellers(t))
            buyer ≜ RandomElement(Buyers)
            price_range ≜ 0 . . (INIT_XTZ ÷ INIT_TOKEN)
            price ≜ RandomElement(price_range)

            MakeBuy(b, p) ≜
              LET xtz_amount ≜ RandomElement(0 . . xtzMap[b])
              IN  CreateBuyingOrder(t, b, p, xtz_amount)

            ExecuteBuy(s) ≜
              LET matches ≜ {x ∈ orders : x.key[3] = TRUE}
                  token_amount ≜ RandomElement(0 . . tokenMap[t][s])
              IN
              IF matches ≠ {}
               THEN ExecuteBuyingOrder(Pick(matches), s, token_amount)
               ELSE FALSE

            MakeSell(s, p) ≜
              LET token_amount ≜ RandomElement(0 . . tokenMap[t][s])
              IN  CreateSellingOrder(t, s, p, token_amount)

            ExecuteSell(b) ≜
              LET matches ≜ {x ∈ orders : x.key[3] = FALSE}
                  xtz_amount ≜ RandomElement(0 . . xtzMap[b])
              IN
              IF matches ≠ {}
            THEN ExecuteSellingOrder(Pick(matches), b, xtz_amount)
            ELSE  FALSE

      IN
      LET BuyerOp  ≜  ∧ Buyers ≠ {}
                      ∧ ∨ MakeBuy(buyer, price)
                        ∨ ExecuteSell(buyer)

          SellerOp ≜  ∧ Sellers(t) ≠ {}
                      ∧ ∨ ExecuteBuy(seller)
                        ∨ MakeSell(seller, price)

      IN  ∨ BuyerOp
          ∨ SellerOp

 IN   Inside(token)
```