```
prod : Nat → Nat → Nat
prod = λm:Nat. λn:Nat.
        m <Nat> (suma n) zero



modulo : List Nat → Nat
modulo = λns:List Nat.
          sort (
                ns <Nat>
                   (λhead:Nat. λrest:Nat. suma (prod head head) rest)
                   zero
                )
```

a) instance (Monad m) ⇒ (MEstado m s) where

return a = Es (λs → return$_m$ (a, s))

m >>= f = Es (λs → let mas = runSt m s
in mas >>=$_m$ (λ(a, s') → runSt (f a) s'))

Nota: Uso un subíndice m para señalar las instancias de return y >>= de la mónada

b) falla :: Parser a
falla = Es (λs → [])

item :: Parser Char
item = Es (λs → case s of
[] → []
(x:xs) → [(x, xs)])

c) data BaseType = TInt | TChar | TBool
data Type = TList BaseType | TFun BaseType Type

parseTy :: Parser Type
parseTy = parseList <+> parseFun

parseList :: Parser Type
parseList = do cadena "["
type ← parseBase
cadena "]"
return (TList type)

parseFun :: Parser Type
parseFun = do btype ← parseBas
cadena "→"
type ← parseTy
return (Fun btype type

```
parseBase :: Parser BaseType
parseBase = parseInt <+> parseChar <+> parseBool

parseInt :: Parser BaseType
parseInt = do cadena "Int"
              return TInt
parseChar :: Parser BaseType
parseChar = do cadena "Char"
               return TChar
parseBool :: Parser BaseType
parseBool = do cadena "Bool"
               return TBool
```

a) $$\frac{\langle c_1, \sigma\rangle \Downarrow \langle ok, \sigma'\rangle \quad \langle c_2, \sigma'\rangle \Downarrow \langle nf, \sigma''\rangle}{\langle c_1; c_2, \sigma\rangle \Downarrow \langle nf, \sigma''\rangle}\ \text{E-SEQ-OK}$$

$$\frac{\langle c_1, \sigma\rangle \Downarrow \langle exc\ e, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \Downarrow \langle exc\ e, \sigma'\rangle}\ \text{E-SEQ-EXC}$$

$$\frac{\langle a, \sigma\rangle \Rightarrow v}{\langle x := a, \sigma\rangle \Downarrow \langle ok, \sigma[x \mapsto v]\rangle}\ \text{E-ASS}$$

$$\frac{\langle a, \sigma\rangle \Rightarrow false \quad \langle c', \sigma\rangle \Downarrow \langle nf, \sigma'\rangle}{\langle if\ a\ c\ c', \sigma\rangle \Downarrow \langle nf, \sigma'\rangle}\ \text{E-IFF}$$

$$\frac{\langle a, \sigma\rangle \Rightarrow true \quad \langle c, \sigma\rangle \Downarrow \langle nf, \sigma'\rangle}{\langle if\ a\ c\ c', \sigma\rangle \Downarrow \langle nf, \sigma'\rangle}\ \text{E-IFT}$$

$$\frac{\langle a, \sigma\rangle \Rightarrow false}{\langle while\ a\ c, \sigma\rangle \Downarrow \langle ok, \sigma\rangle}\ \text{E-WHILEF}$$

$$\frac{\langle a, \sigma\rangle \Rightarrow true \quad \langle c, \sigma\rangle \Downarrow \langle ok, \sigma'\rangle \quad \langle while\ a\ c, \sigma'\rangle \Downarrow \langle nf, \sigma''\rangle}{\langle while\ a\ c, \sigma\rangle \Downarrow \langle nf, \sigma''\rangle}\ \text{E-WHILET-OK}$$

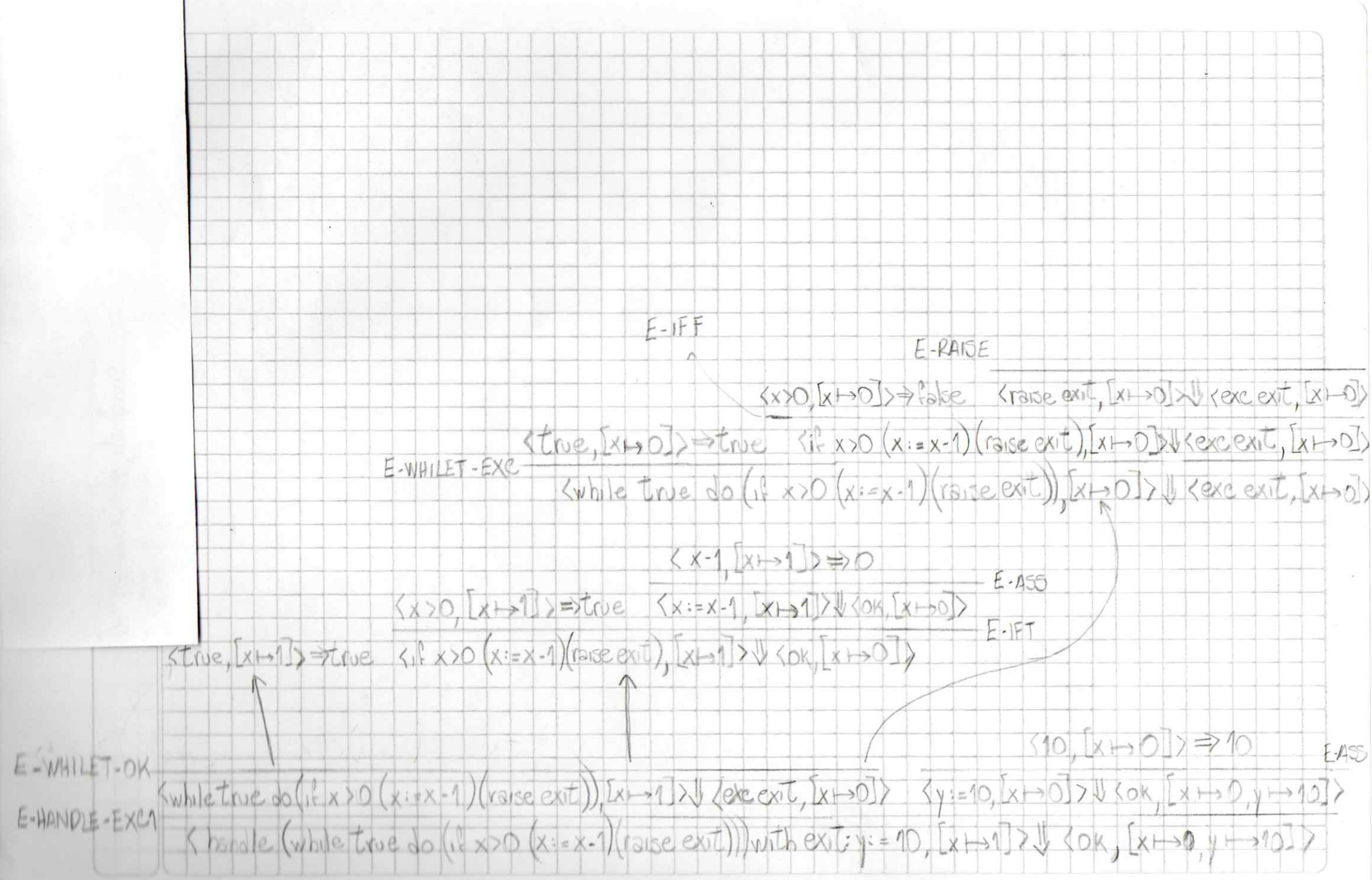
$$\frac{\langle a, \sigma\rangle \Rightarrow true \quad \langle c, \sigma\rangle \Downarrow \langle exc\ e, \sigma'\rangle}{\langle while\ a\ c, \sigma\rangle \Downarrow \langle exc\ e, \sigma'\rangle}\ \text{E-WHILET-EXC}$$

$$\frac{}{\langle raise\ e, \sigma\rangle \Downarrow \langle exc\ e, \sigma\rangle}\ \text{E-RAISE}$$

$$\frac{\langle c, \sigma\rangle \Downarrow \langle ok, \sigma'\rangle}{\langle handle\ c\ with\ e : c', \sigma\rangle \Downarrow \langle ok, \sigma'\rangle}\ \text{E-HANDLE-OK}$$

$$\frac{\langle c, \sigma \rangle \Downarrow \langle \text{exc } e, \sigma' \rangle \qquad \langle c', \sigma' \rangle \Downarrow \langle nf^{?}, \sigma'' \rangle}{\langle \text{handle } c \text{ with } e : c', \sigma \rangle \Downarrow \langle nf^{?}, \sigma'' \rangle} \text{ E-HANDLE-EXC1}$$

$$\frac{\langle c, \sigma \rangle \Downarrow \langle \text{exc } e', \sigma' \rangle \qquad e' \neq e}{\langle \text{handle } c \text{ with } e : c', \sigma \rangle \Downarrow \langle \text{exc } e', \sigma' \rangle} \text{ E-HANDLE-EXC2}$$


E-IFF
E-RAISE
⟨x>0, [x↦0]⟩ ⇒ false
⟨raise exit, [x↦0]⟩ ⇓ ⟨exc exit, [x↦0]⟩
E-WHILET-EXC
⟨true, [x↦0]⟩ ⇒ true
⟨if x>0 (x:=x-1)(raise exit), [x↦0]⟩ ⇓ ⟨exc exit, [x↦0]⟩
⟨while true do (if x>0 (x:=x-1)(raise exit)), [x↦0]⟩ ⇓ ⟨exc exit, [x↦0]⟩
⟨x-1, [x↦1]⟩ ⇒ 0
E-ASS
⟨x>0, [x↦1]⟩ ⇒ true
⟨x:=x-1, [x↦1]⟩ ⇓ ⟨ok, [x↦0]⟩
E-IFT
⟨true, [x↦1]⟩ ⇒ true
⟨if x>0 (x:=x-1)(raise exit), [x↦1]⟩ ⇓ ⟨ok, [x↦0]⟩
E-WHILET-OK
⟨while true do (if x>0 (x:=x-1)(raise exit)), [x↦1]⟩ ⇓ ⟨exc exit, [x↦0]⟩
⟨10, [x↦0]⟩ ⇒ 10
E-ASS
⟨y:=10, [x↦0]⟩ ⇓ ⟨ok, [x↦0, y↦10]⟩
E-HANDLE-EXC1
⟨handle (while true do (if x>0 (x:=x-1)(raise exit))) with exit: y:=10, [x↦1]⟩ ⇓ ⟨ok, [x↦0, y↦10]⟩