NaturalSemantics.pdf

Juanma21_

Teoría de Los Lenguajes de Programación

4º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                  <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

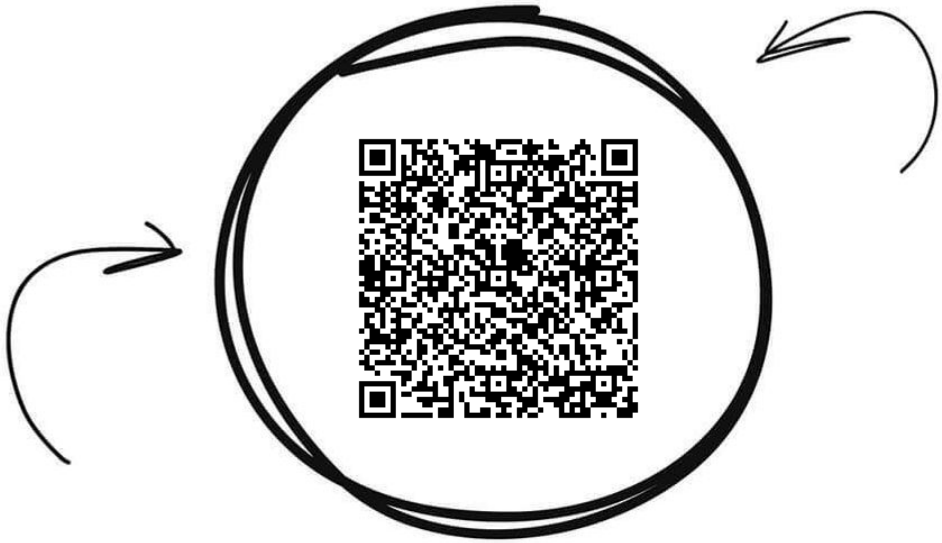
Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

Teoría de Los Lenguajes de P...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                  <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell

-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23

```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                 <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                 <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                 <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                  <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```

NaturalSemantics.hs

```haskell
-- ------------------------------------------------------------------
-- Natural Semantics for WHILE 2020.
-- Examen de Lenguajes de Programación. UMA.
--
-- Apellidos, Nombre: Cardeñosa Borrego, Juan Manuel
-- ------------------------------------------------------------------

module NaturalSemantics where

import           While20

updateState :: State -> Var -> Z -> State
updateState s x v y = if x == y then v else s y

-- representation of configurations for While

data Config = Inter Stm State  -- <S, s>
            | Final State      -- s

-- representation of the transition relation <S, s> -> s'

nsStm :: Config -> Config

-- x := a

nsStm (Inter (Ass x a) s) = Final (updateState s x (aVal a s))

-- skip

nsStm (Inter Skip s) = Final s

-- s1; s2

nsStm (Inter (Comp ss1 ss2) s) = Final s''
  where
    Final s'  = nsStm (Inter ss1 s)
    Final s'' = nsStm (Inter ss2 s')

-- if b then s1 else s2

nsStm (Inter (If b ss1 ss2) s)
  | bVal b s = Final s'
  where
    Final s' = nsStm (Inter ss1 s)

nsStm (Inter (If b ss1 ss2) s)
  | not(bVal b s) = Final s'
  where
    Final s' = nsStm (Inter ss2 s)

-- ------------------------------------------------------------------
```

```haskell
-- NO MODIFICAR EL CODIGO DE ARRIBA
-----------------------------------------------------------------------

{-
                          <x := y, s> -> s'
 [swap ns] ------------------------------------------------
                <Swap x y, s> -> s'[y -> A[x]s]
-}

-- Sentencia Swap
nsStm (Inter (Swap x y) s) = Final (updateState s1 y (s x))
  where
    Final s1 = nsStm (Inter (Ass x (V y)) s)

{-

Si es True
              <s1, s> -> s' <s3, s'> -> s''  <s2, s''> -> s'''  <For Skip b s2 s3, s'''> ->
s''''
 [for ns] ------------------------------------------------------------------------------
----- if B[b]s = True
                                                  <For s1 b s2 s3, s> -> s''''

Si es False
                       <s1,s> -> s'
  [for ns] ------------------------------------ if B[b]s = False
                <For s1 b s2 s3, s> -> s'

-}

-- Sentencia for
-- Si es true
nsStm(Inter (For s1 b s2 s3) s) | bVal b s = Final s''''
  where
    Final s'    = nsStm (Inter s1 s)
    Final s''   = nsStm (Inter s3 s')
    Final s'''  = nsStm (Inter s2 s'')
    Final s'''' = nsStm (Inter (For Skip b s2 s3) s''')

-- Si es false
nsStm(Inter (For s1 b s2 s3) s) | not(bVal b s) = Final s'
  where
    Final s'    = nsStm (Inter s1 s)


reduce :: Aexp -> Aexp

-- Entero
reduce (N n) = (N n)

-- Variable
reduce (V x) = (V x)

-- Suma 1
```

```haskell
reduce (Add (N n1) (N n2)) = reduce (N (n1 + n2))

-- Suma 2
reduce (Add a1 a2) = Add (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Resta 1
reduce (Sub (N n1) (N n2)) = reduce (N (n1 - n2))

-- Resta 2
reduce (Sub a1 a2) = Sub (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


-- Mult 1
reduce (Mult (N n1) (N n2)) = reduce (N (n1 * n2))

-- Mult 2
reduce (Mult a1 a2) = Mult (reduce a1') (reduce a2')
  where
    a1' = reduce a1
    a2' = reduce a2


exp1 :: Aexp
exp1 = Mult (Add (Add (N 3) (N 6)) (V "x")) (Sub (Mult (N 4) (N 8)) (V "y"))
-- exp1 = ((3 + 6) + (x)) * ((4 * 8) - y)
-- reduce exp1
-- Solucion: (9 + x) * (32 - y)

exp2 :: Aexp
exp2 = Add (V "x") (Mult (N 5) (N 3))
-- exp2 = x + (5 * 3)
-- reduce exp2
-- Solucion: x + 15

exp3 :: Aexp
exp3 = Add (Mult (N 8) (V "y")) (Add (Mult (N 3) (N 2)) (N 5))
-- exp3 = (8* y) + ((3 + 2) * 5)
-- reduce exp3
-- Solucion: (8 * y) + 11

exp4 :: Aexp
exp4 = Add (Mult (N 8) (V "y")) (Add (Mult (Mult (N 4) (Mult (N 2) (Mult (N 3) (N 4)))) (N 2)) (N 5))
-- exp4 = (8 * y) + ((4 * (2 * (3 * 2))) + 5)
-- reduce exp4
-- Solucion: (8 * y) + 197

```

```haskell
-- Swap como funcion para probar, se pide el stm (arriba está hecho)
s1 :: State
s1 "x" = 3
s1 "y" = 4

exp5 :: Aexp
exp5 = Add (Mult (V "x") (N 5) ) (V "y")

-- aVal exp5 s1
-- Da 19
-- aVal exp5 (swap s1 "x" "y")
-- Da 23
```