Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Compiladores

Optimizando la Máquina Virtual

Ejercicio opcional 7

Santucci, Tomas
Cavagna, Lucas Gastón

Año 2024

Demostración

Teorema principal

Primero vamos a dejar la estructura del árbol sobre el cual vamos a aplicar inducción estructural:

```
data Tm info var =
    V info var
  | Const info Const
  | Lam info Name Ty (Scope info var)
  | App info (Tm info var) (Tm info var)
  | Print info String (Tm info var)
  | BinaryOp info BinaryOp (Tm info var) (Tm info var)
  | Fix info Name Ty Name Ty (Scope2 info var)
  | IfZ info (Tm info var) (Tm info var) (Tm info var)
  | Let info Name Ty (Tm info var) (Scope info var)

type TTerm = Tm (TermInfo,Ty) Var
```

Las funciones que intervienen en la demostración son: bcc (versión sin optimizaciones), bcc (con optimizaciones), length y bct. Aquí solo vamos a dejar el código de la versión sin optimizaciones de bcc ya que las demás funciones pueden ser consultadas en el código del archivo Bytecompile.hs (salvo length que esta disponible en el preludio de Haskell):

```
bcc :: MonadFD4 m => TTerm -> m Bytecode
bcc (V _ (Bound i)) = return [ACCESS,i]

bcc (Const _ (CNat i)) = return [CONST,i]

bcc (Lam _ _ _ (Sc1 t)) = do
  ct <- bcc t
  return $ [FUNCTION, (length ct) + 1] ++ ct ++ [RETURN]

bcc (App _ t1 t2) = do
  ct1 <- bcc t1
  ct2 <- bcc t2
  return $ ct1 ++ ct2 ++ [CALL]

bcc (Print _ str t) = do
  ct <- bcc t
  return $ ct ++ [PRINT] ++ (string2bc str) ++ [NULL,PRINTN]

bcc (BinaryOp _ op t1 t2) = do
  ct1 <- bcc t1
  ct2 <- bcc t2
  return $ ct1 ++ ct2 ++ (op2bc op)

bcc (Fix _ _ _ _ _ (Sc2 t)) = do
  ct <- bcc t
  return  $ [FUNCTION, (length ct) + 1] ++ ct ++ [RETURN,FIX]

bcc (IfZ _ c t e) = do
  cc <- bcc c
```

```
  ct <- bcc t
  ce <- bcc e
  return $ cc ++ [CJUMP, (length ct) + 2] ++ ct ++ [JUMP,length ce] ++ ce

bcc (Let _ _ _ def (Sc1 body)) = do
  cdef <- bcc def
  cbody <- bcc body
  return $ cdef ++ [SHIFT] ++ cbody ++ [DROP]

bcc t = failFD4 (show t)
```

Definición (Inducción estructural para TTerm)

Dada una propiedad P sobre elementos de TTerm, para probar $\forall t$ :: TTerm. P(t) :

- Probamos para *P(V info var)* y *P(Const info Const)*
- Probamos que si *P(t_1)* entonces *P(Lam info Name Ty (Sc1 t_1))*
  , *P(Print info String t_1)*
  y *P(Fix info Name Ty Name Ty (Sc2 t_1))*
  $\wedge$ si *P(t_1)* y *P(t_2)* entonces *P(App info t_1 t_2)*
  , *P(BinaryOp info BinaryOp t_1 t_2)*
  y *P(Let info Name Ty t_1 (Sc1 t_2))*
  $\wedge$ si *P(t_1)*, *P(t_2)* y *P(t_3)* entonces *P(IfZ info t_1 t_2 t_3)*
- Probaremos que $\forall t$ :: TTerm. *length(bcc_V(t))* $\geq$ *length(bcc_N(t))*

Nota: Se aclara que bcc_N hace referencia a la función bcc con las optimizaciones ya implementadas y bcc_V a la versión anterior sin optimizaciones.
Ahora procedemos con la demostración:

DEM)

- Casos bases
  - Caso *V info var*
    length(bcc_V(V _ var))
    $= \langle$*bcc_V.V*$\rangle$
    length([ACCESS ,i_{var}])
    $= \langle$*length. 1* $\wedge$ *length.2* $\rangle$
    2
    Por otro lado
    length(bcc_N(V _ var))
    $= \langle$*bcc_N.V*$\rangle$
    length([ACCESS ,i_{var}])
    $= \langle$*length. 1* $\wedge$ *length.2* $\rangle$
    2
  - Caso *Const info Const*
    Análogo al caso anterior
- Paso inductivo

- Caso *IfZ info* t_1 t_2 t_3

  H.I.i = *length(bcc_V(t_i))* $\geq$ *length(bcc_N(t_i)) con i = 1..3*

  length(bcc_V(IfZ _ t_1 t_2 t_3 *))*
  = $\langle$*bcc_V.IfZ*$\rangle$
  length(bcc_V(t_1 *) ++ [CJUMP, n1] ++ bcc_V(* t_2 *) ++ [JUMP,n2] ++ bcc_V(* t_3 *))*
  = $\langle$*Lema2 X4*$\rangle$
  length(bcc_V(t_1 *)) + length([CJUMP, n1]) + length(bcc_V(* t_2 *)) + length([JUMP,n2]) + length(bcc_V(* t_3 *))*
  $\geq$ $\langle$*H.I.i* con i = 1..3 $\rangle$
  length(bcc_N(t_1 *)) + length([CJUMP, n1]) + length(bcc_N(* t_2 *)) + length([JUMP,n2])+ length(bcc_N(* t_3 *))*
  = $\langle$*Lema2 X4*$\rangle$
  length(bcc_N(t_1 *) ++ [CJUMP, n1] ++ bcc_N(* t_2 *) ++ [JUMP,n2] ++ bcc_N(* t_3*))*
  = $\langle$*bcc_N.IfZ*$\rangle$
  length(bcc_N(IfZ _ t_1 t_2 t_3 *))*

- Caso *Print info String* t_1
  Análogo al caso del IfZ

- Caso *App info* t_1 t_2
  Análogo al caso del IfZ

- Caso *BinaryOp info BinaryOp* t_1 t_2
  Análogo al caso del IfZ

- Caso *Let info Name Ty* t_1 *(Sc1* t_2*)*
  Análogo al caso del IfZ

- Caso *Lam info Name Ty (Sc1* t_1*)*

  H.I = *length(bcc_V(t_1))* $\geq$ *length(bcc_N(t_1))*

  length(bcc_V(Lam info Name Ty (Sc1 t_1*))*
  = $\langle$*bcc_V.Lam*$\rangle$
  length([FUNCTION ,n1]++ bcc_V(t_1) ++ [RETURN])
  = $\langle$*Lema2 X3*$\rangle$
  length([FUNCTION ,n1]) + length(bcc_V(t_1)) + length([RETURN])
  $\geq$ $\langle$*H.I* $\rangle$
  length([FUNCTION ,n1]) + length(bcc_N(t_1)) + length([RETURN])
  = $\langle$ length.1 $\wedge$ length.2 $\rangle$
  length([FUNCTION ,n1]) + length(bcc_N(t_1)) + 1
  $\geq$ $\langle$ Lema 1 $\rangle$
  length([FUNCTION ,n1]) + length(bct(t_1))
  = $\langle$*Lema2* $\rangle$
  length([FUNCTION ,n1]++ bct(t_1))
  = $\langle$*bcc_N.Lam*$\rangle$
  length(bcc_N(Lam info Name Ty (Sc1 t_1*))*

- Caso *Fix info Name Ty Name Ty (Sc2* t_1*)*
  Análogo al caso del Lam

Lema 1

Definición (Inducción estructural para TTerm)
Dada una propiedad Q sobre elementos de TTerm, para probar $\forall t$:: TTerm. Q(t) :

- Probamos para *Q(V info var)* y *P(Const info Const)*
- Probamos que si $Q(t_1)$ entonces $Q(Lam\ info\ Name\ Ty\ (Sc1\ t_1))$
  , $Q(Print\ info\ String\ t_1)$
  y $Q(Fix\ info\ Name\ Ty\ Name\ Ty\ (Sc2\ t_1))$
  $\wedge$ si $Q(t_1)$ y $Q(t_2)$ entonces $Q(App\ info\ t_1\ t_2)$
  , $Q(BinaryOp\ info\ BinaryOp\ t_1\ t_2)$
  y $Q(Let\ info\ Name\ Ty\ t_1\ (Sc1\ t_2))$
  $\wedge$ si $Q(t_1)$, $Q(t_2)$ y $Q(t_3)$ entonces $Q(IfZ\ info\ t_1\ t_2\ t_3)$
- Probaremos que $\forall t$ :: TTerm. *length(bcc_N(t)) + 1 $\geq$ length(bct(t))*
- Casos bases
  - Caso *V info var*
    length(bcc_N(V _ var)) + 1
    = ⟨*bcc_N.V*⟩
    length([ACCESS ,i_{var}]) + 1
    = ⟨*length. 1 $\wedge$ length.2* ⟩
    3

    Por otro lado
    length(bct(V _ var))
    = ⟨*bct.N*⟩
    length(bcc_N(V _ var))
    = ⟨*bcc_N.V*⟩
    length([ACCESS ,i_{var}])
    = ⟨*length. 1 $\wedge$ length.2* ⟩
    2
  - Caso *Const info Const*
    Análogo al caso anterior
- Paso inductivo
  - Caso *IfZ info t_1 t_2 t_3*

    H.I.i = length(bcc_N(t_i)) + 1 $\geq$ length(bct(t_i)) con i = 1..3

    length(bcc_N(IfZ info t_1 t_2 t_3)) + 1
    = ⟨*bcc_N.IfZ*⟩
    length(bcc_N(t_1) ++ [CJUMP, n1] ++ bcc_N(t_2) ++ [JUMP,n2]++ bcc_N(t_3)) + 1
    = ⟨*Lema2 X4*⟩
    length(bcc_N(t_1)) + length([CJUMP, n1]) + length(bcc_N(t_2)) + length([JUMP,n2])+ length(bcc_N(t_3)) + 1
    = ⟨ length.1 $\wedge$ length.2 ⟩
    length(bcc_N(t_1)) + length([CJUMP, n1]) + length(bcc_N(t_2)) + length(bcc_N(t_3)) + 3
    $\geq$ ⟨*H.I.i* con i = 2..3 ⟩

$length(bcc\_N(t_1)) + length([CJUMP, n1]) + length(bct(t_2)) + length(bct(t_3))$
$= \langle Lema2\ X4 \rangle$
$length(bcc\_N(t_1)\ {+}{+}\ [CJUMP, n1]\ {+}{+}\ bct(t_2)\ {+}{+}\ bct(t_3))$
$= \langle bct.IfZ \rangle$
$length(bct(IfZ\ info\ t_1\ t_2\ t_3))$

- Caso *Let info Name Ty* t_1 *(Sc1* t_2*)*

  $H.I.i = length(bcc\_N(t_i)) + 1 \geq length(bct(t_i))\ con\ i = 1..2$

  $length(bcc\_N(Let\ info\ Name\ Ty\ t_1\ (Sc1\ t_2))) + 1$
  $= \langle bcc\_N.Let \rangle$
  $length(bcc\_N(t_1)\ {+}{+}\ [SHIFT]\ {+}{+}\ bcc\_N(t_2)\ {+}{+}\ [DROP]) + 1$
  $= \langle Lema2\ X3 \rangle$
  $length(bcc\_N(t_1)) + length([SHIFT]) + length(bcc\_N(t_2)) + length([DROP]) + 1$
  $= \langle length.\ 1 \wedge length.2\ \rangle$
  $length(bcc\_N(t_1)) + length([SHIFT]) + length(bcc\_N(t_2)) + 2$
  $\geq \langle H.I.2 \rangle$
  $length(bcc\_N(t_1)) + length([SHIFT]) + length(bct(t_2))$
  $= \langle Lema2X3 \rangle$
  $length(bcc\_N(t_1)\ {+}{+}\ [SHIFT]\ {+}{+}\ bct(t_2))$
  $= \langle bct.Let \rangle$
  $length(bct(Let\ info\ Name\ Ty\ t_1\ (Sc1\ t_2))$

- Caso *App info* t_1 t_2
  $H.I.i = length(bcc\_N(t_i)) + 1 \geq length(bct(t_i))\ con\ i = 1..2$

  $length(bcc\_N(App\ info\ t_1\ t_2) + 1$
  $\geq \langle\rangle$
  $length(bcc\_N(App\ info\ t_1\ t_2)$
  $= \langle bcc\_N.App \rangle$
  $length(bcc\_N(t_1){+}{+}\ bcc\_N(t_2){+}{+}\ [CALL])$
  $= \langle Lema2\ X2\ \rangle$
  $length(bcc\_N(t_1) + length(bcc\_N(t_2)) + length([CALL])$
  $= \langle length.\ 1 \wedge length.2\ \rangle$
  $length(bcc\_N(t_1) + length(bcc\_N(t_2)) + 1$
  $= \langle length.\ 1 \wedge length.2\ \rangle$
  $length(bcc\_N(t_1) + length(bcc\_N(t_2)) + length([TAILLCALL])$
  $= \langle Lema2\ X2\ \rangle$
  $length(bcc(t_1){+}{+}\ bcc(t_2){+}{+}\ [TAILCALL])$
  $= \langle bcc\_N.App \rangle$
  $length(bct(App\ info\ t_1\ t_2))$

- Casos restantes (término t):

  $length(bcc\_N(t)) + 1$
  $= \langle\ \text{length.1} \wedge \text{length.2}\ \rangle$
  $length(bcc\_N(t)) + length([RETURN])$
  $= \langle Lema2\ \rangle$
  $length(bcc\_N(t)\ {+}{+}\ [RETURN])$
  $= \langle bct.t \rangle$
  $length(bct(t))$

Lema 2

Definición (Inducción estructural para listas)
Dada una propiedad Q sobre listas ,para probar $\forall xs$:: [a]. T(t) :

- Probamos para *T([])*
- Probamos que si *Q(xs)* entonces *Q(x:xs)*
- Probamos que $\forall xs$ :: [a]. *length(xs ++ ys) = length(xs)+length(ys)*
- Caso base
  - Caso []
    length([]++ys)
    = ⟨++.1 ⟩
    length(ys)
    = ⟨aritmética ⟩
    0 + length(ys)
    = ⟨length.1 ⟩
    length([]) + length(ys)
  - Caso x:xs
    H.I = length(xs ++ ys) = length(x:xs)+length(ys)

    length((x:xs) ++ ys)
    = ⟨++.2 ⟩
    length(x:(xs ++ ys))
    = ⟨length.2 ⟩
    1+length(xs ++ ys)
    = ⟨H.I. ⟩
    1+length(xs) + length(ys)
    = ⟨length.2 ⟩
    length(x:xs) + length(ys)