Demostración Compiladores

Camila Koatz - Yamil Saer

Marzo 2024

Demostración

Vamos a demostrar que la compilación a bytecode con la función optimizada no genera una código mas grande que la versión sin optimizar.

Sea *bcc* la versión sin optimizaciones del compilador a bytecode y *bcc_o* la versión optimizada.

La estructura a compilar tiene la siguiente forma:

```
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 (Pos,Ty) Var
```

Vamos a realizar inducción estructural sobre TTerm para probar lo siguiente:

$$\forall t :: TTerm. \quad length(bcc(t)) \geq length(bcc\_o(t))$$

- **Casos base:**
  - $t = V\ i\ var$

    $length(bcc(V\ i\ var))$
    = (def. bcc.V)
    $length([ACCESS, n_{var}])$
    = (def. bcc_o.V)
    $length(bcc\_o(V\ i\ var))$

* t = *Const i c*

  $length(bcc(Const\ i\ c))$
  $=$ (def. bcc.Const)
  $length([CONST, n_c])$
  $=$ (def. bcc_o.Const)
  $length(bcc\_o(V\ i\ var))$

- **Casos inductivos:**

  * t = *App i* t_1 t_2

    Hipótesis inductiva:

    $$H1)\quad t_1 :: TTerm.\quad length(bcc(t_1)) \geq length(bcc\_o(t_1))$$

    $$H2)\quad t_2 :: TTerm.\quad length(bcc(t_2)) \geq length(bcc\_o(t_2))$$

    $length(bcc(App\ i\ t_1\ t_2))$
    $=$ (def. bcc.App)
    $length(bcc(t_1) \mathbin{++} bcc(t_2) \mathbin{++} [CALL])$
    $=$ (Lema 1)
    $length(bcc(t_1)) + length(bcc(t_2)) + length([CALL])$
    $\geq$ (H1 y H2)
    $length(bcc\_o(t_1)) + length(bcc\_o(t_2)) + length([CALL])$
    $=$ (Lema 1)
    $length(bcc\_o(t_1)) \mathbin{++} bcc\_o(t_2) \mathbin{++} [CALL])$
    $=$ (def. bcc_o.App) $length(bcc\_o(App\ i\ t_1\ t_2))$

  * t = *Print i str t'*. Análogo al caso App.

  * t = *BinaryOp i op* t_1 t_2. Análogo al caso App.

  * t = *IfZ i c* t_1 t_2. Análogo al caso App.

  * t = *Let i n ty* t_1 *(Sc1* t_2*)*. Análogo al caso App.

  * t = *Lam i n ty (Sc1 t')*

    Hipótesis inductiva:

    $$t' :: TTerm.\quad length(bcc(t')) \geq length(bcc\_o(t'))$$

    $length(bcc(Lam\ i\ n\ ty\ (Sc1\ t'))$
    $=$ (def. bcc.Lam)
    $length([FUNCTION, n_{t'}] \mathbin{++} bcc(t') \mathbin{++} [RETURN])$
    $=$ (Lema 1)
    $length([FUNCTION, n_{t'}]) + length(bcc(t')) + length([RETURN])$
    $\geq$ (H.I.)
    $length([FUNCTION, n_{t'}]) + length(bcc\_o(t')) + length([RETURN])$
    $=$ (def. length)
    $length([FUNCTION, n_{t'}]) + length(bcc\_o(t')) + 1$
    $\geq$ (Lema 2)

$length([FUNCTION, n_{t'}]) + length(bct(t'))$
$=$ (Lema 1)
$length([FUNCTION, n_{t'}] ++ bct(t'))$
$=$ (def. bcc_o.Lam)
$length(bcc\_o(Lam\ i\ n\ ty\ (Sc1\ t')))$

* $t = Fix\ i\ n_1\ ty_1\ n_2\ ty_2\ (Sc2\ t')$.Análogo al caso Lam.

Demostración Lema 1

Vamos a probar por inducción estructural sobre List la siguiente propiedad:

$$\forall xs, ys :: [a]. \quad length(xs ++ ys) = length(xs) + length(ys)$$

- **Caso base:** $xs = [\ ]$

  $length([\ ] ++ ys)$
  $=$ (def. (++).1)
  $length(ys)$
  $= 0 + length(ys)$
  $=$ (def. length.1)
  $length([\ ]) + length(ys)$

- **Caso inductivo:** $xs = x : xs'$

  Hipótesis inductiva:

  $$xs', ys :: [a]. \quad length(xs' ++ ys) = length(xs') + length(ys)$$

  $length((x : xs) ++ ys)$
  $=$ (def. (++).2)
  $length(x : (xs' ++ ys))$
  $=$ (def. length.2)
  $1 + length(xs' ++ ys)$
  $=$ (H.I.)
  $1 + length(xs') + length(ys)$
  $=$ (def. length.2)
  $length(x : xs') + length(ys)$

Demostración Lema 2

Vamos a probar por inducción estructural sobre TTerm la siguiente propiedad:

$$\forall t :: TTerm. \quad length(bcc\_o(t)) + 1 \geq length(bct(t))$$

- **Casos base:**

  $*$ $t = V\ i\ var$

  $length(bcc\_o(V\ i\ var)) + 1$
  $=$ (def. length)
  $length(bcc\_o(V\ i\ var)) + length([RETURN])$
  $=$ (def. bct.V)
  $length(bct(V\ i\ var))$

  $*$ $t = Const\ i\ c$

  $length(bcc\_o(Const\ i\ c)) + 1$
  $=$ (def. length)
  $length(bcc\_o(Const\ i\ c)) + length([RETURN])$
  $=$ (def. bct.Const)
  $length(bct(Const\ i\ c))$

- **Casos inductivos:**

  $*$ $t = App\ i\ t_1\ t_2$

  $length(bcc\_o(App\ i\ t_1\ t_2)) + 1$
  $\geq length(bcc\_o(App\ i\ t_1\ t_2)$
  $=$ (def. bcc_o.App)
  $length(bcc\_o(t_1) +\!\!+ bcc\_o(t_2) +\!\!+ [CALL])$
  $=$ (Lema 1)
  $length(bcc\_o(t_1)) + length(bcc\_o(t_2)) + length([CALL])$
  $=$ (def. length)
  $length(bcc\_o(t_1)) + length(bcc\_o(t_2)) + 1$
  $=$ (def. length)
  $length(bcc\_o(t_1)) + length(bcc\_o(t_2)) + length([TAILCALL])$
  $=$ (Lema 1)
  $length(bcc\_o(t_1) +\!\!+ bcc\_o(t_2) +\!\!+ [TAILCALL])$
  $=$ (def. bct.App)
  $length(bct\_o(App\ i\ t_1\ t_2))$

  $*$ $t = IfZ\ i\ c\ t_1\ t_2$

  Hipótesis inductiva:

  $$H1) \quad t_1 :: TTerm. \quad length(bcc\_o(t_1)) + 1 \geq length(bct(t_1))$$

  $$H2) \quad t_2 :: TTerm. \quad length(bcc\_o(t_2)) + 1 \geq length(bct(t_2))$$

  $length(bcc\_o(IfZ\ i\ c\ t_1\ t_2)) + 1$
  $=$ (def. bcc_o.IfZ)

$length(bcc\_o(c) +\!+ [CJUMP, n_1] +\!+ bcc\_o(t_1) +\!+ [JUMP, n_2] +\!+ bcc\_o(t_2)) + 1$
$=$ (Lema 1)
$length(bcc\_o(c)) + length([CJUMP, n_1]) + length(bcc\_o(t_1))$
$+length([JUMP, n_2]) + length(bcc\_o(t_2)) \ + \ 1$
$=$ (def. length)
$length(bcc\_o(c)) + length([CJUMP, n_1]) + length(bcc\_o(t_1)) + length(bcc\_o(t_2)) + 3$
$\geq$ (H1 y H2)
$length(bcc\_o(c)) + length([CJUMP, n_1]) + length(bct(t_1)) + length(bct(t_2))$
$=$ (Lema 1)
$length(bcc\_o(c) +\!+ [CJUMP, n_1] +\!+ bct(t_1) +\!+ bct(t_2))$
$=$ (def. bct.IfZ)
$length(bct(IfZ\ i\ c\ t_1\ t_2))$

* $t = Let\ i\ n\ ty\ t_1\ (Sc1\ t_2)$

  Hipótesis inductiva:

  $$t_2 :: TTerm. \quad length(bcc\_o(t_2)) + 1 \geq length(bct(t_2))$$

  $length(bcc\_o(Let\ i\ n\ ty\ t_1\ (Sc1\ t_2))) + 1$
  $=$ (def. bcc_o.Let)
  $length(bcc\_o(t_1) +\!+ [SHIFT] +\!+ bcc\_o(t_2) +\!+ [DROP])$
  $=$ (Lema 1)
  $length(bcc\_o(t_1)) + length([SHIFT]) + length(bcc\_o(t_2)) + length([DROP])$
  $=$ (def. length)
  $length(bcc\_o(t_1)) + length([SHIFT]) + length(bcc\_o(t_2)) + 1$
  $\geq$ (H.I.)
  $length(bcc\_o(t_1)) + length([SHIFT]) + length(bct(t_2))$
  $=$ (Lema 1)
  $length(bcc\_o(t_1) +\!+ [SHIFT] +\!+ bct(t_2))$
  $=$ (def. bct.Let)
  $length(bct(Let\ i\ n\ ty\ t_1\ (Sc1\ t_2)))$

* Casos restantes: $t :: TTerm$

  $length(bcc\_o(t)) + 1$
  $=$ (def. length)
  $length(bcc\_o(t)) + length([RETURN])$
  $=$ (Lema 1)
  $length(bcc\_o(t) +\!+ [RETURN])$
  $=$ (def. bct.t)
  $length(bct(t))$