Capítulo 6

Caso de estudio: Compilación de expresiones

El objetivo de esta sección es contruir un compilador de expresiones genéricas (construidas con variables, constantes y operaciones binarias) a una máquina de pila y demostrar su corrección.

6.1 Las expresiones y el intérprete

Definición 6.1.1. *Las expresiones son las constantes, las variables (representadas por números naturales) y las aplicaciones de operadores binarios a dos expresiones.*

types $'v\ binop = {'v} \Rightarrow {'v} \Rightarrow {'v}$
datatype $'v\ expr =$
  $Const\ 'v$
  $\mid Var\ nat$
  $\mid App\ 'v\ binop\ 'v\ expr\ 'v\ expr$

Definición 6.1.2 (Intérprete)**.** *La función* valor *toma como argumentos una expresión y un entorno (i.e. una aplicación de las variables en elementos del lenguaje) y devuelve el valor de la expresión en el entorno.*

primrec $valor :: {'v}\ expr \Rightarrow (nat \Rightarrow {'v}) \Rightarrow {'v}$ **where**
 $valor\ (Const\ b)\ ent = b$
$\mid valor\ (Var\ x)\ ent = ent\ x$
$\mid valor\ (App\ f\ e1\ e2)\ ent = (f\ (valor\ e1\ ent)\ (valor\ e2\ ent))$

Ejemplo 6.1.3. A continuación mostramos algunos ejemplos de evaluación con el intérprete.

lemma
 valor $(Const\ 3)\ id = 3 \wedge$
 valor $(Var\ 2)\ id = 2 \wedge$
 valor $(Var\ 2)\ (\lambda x.\ x{+}1) = 3 \wedge$
 valor $(App\ (op\ +)\ (Const\ 3)\ (Var\ 2))\ (\lambda x.\ x{+}1) = 6 \wedge$
 valor $(App\ (op\ +)\ (Const\ 3)\ (Var\ 2))\ (\lambda x.\ x{+}4) = 9$
by *simp*

6.2 La máquina de pila

Nota 6.2.1. La máquina de pila tiene tres clases de intrucciones:

- cargar en la pila una constante,
- cargar en la pila el contenido de una dirección y
- aplicar un operador binario a los dos elementos superiores de la pila.

datatype *'v instr* =
  IConst 'v
 | *ILoad nat*
 | *IApp 'v binop*

Definición 6.2.2 (Ejecución). *La ejecución de la máquina de pila se modeliza mediante la función* ejec *que toma una lista de intrucciones, una memoria (representada como una función de las direcciones a los valores, análogamente a los entornos) y una pila (representada como una lista) y devuelve la pila al final de la ejecución.*

primrec *ejec* :: *'v instr list* ⇒ (*nat*⇒*'v*) ⇒ *'v list* ⇒ *'v list* **where**
 ejec [] *ent vs* = *vs*
| *ejec* (*i*#*is*) *ent vs* =
  (*case i of*
   IConst v ⇒ *ejec is ent* (*v*#*vs*)
  | *ILoad x* ⇒ *ejec is ent* ((*ent x*)#*vs*)
  | *IApp f* ⇒ *ejec is ent* ((*f* (*hd vs*) (*hd* (*tl vs*)))#(*tl*(*tl vs*))))

Ejemplo 6.2.3. A continuación se muestran ejemplos de ejecución.

lemma
 ejec [*IConst 3*] *id* [7] = [3,7] ∧
 ejec [*ILoad 2*, *IConst 3*] *id* [7] = [3,2,7] ∧

$\mathit{ejec}\ [\mathit{ILoad}\ 2,\ \mathit{IConst}\ 3]\ (\lambda x.\ x{+}4)\ [7] = [3{,}6{,}7]\ \wedge$
$\mathit{ejec}\ [\mathit{ILoad}\ 2,\ \mathit{IConst}\ 3,\ \mathit{IApp}\ (\mathit{op}\ +)]\ (\lambda x.\ x{+}4)\ [7] = [9{,}7]$
by *simp*

6.3 El compilador

Definición 6.3.1. *El compilador* comp *traduce una expresión en una lista de instrucciones.*

primrec $\mathit{comp} :: {'v}\ \mathit{expr} \Rightarrow {'v}\ \mathit{instr\ list}$ **where**
$\mathit{comp}\ (\mathit{Const}\ v) = [\mathit{IConst}\ v]$
$|\ \mathit{comp}\ (\mathit{Var}\ x) = [\mathit{ILoad}\ x]$
$|\ \mathit{comp}\ (\mathit{App}\ f\ e1\ e2) = (\mathit{comp}\ e2)\ @\ (\mathit{comp}\ e1)\ @\ [\mathit{IApp}\ f]$

Ejemplo 6.3.2. A continuación se muestran ejemplos de compilación.

lemma
$\mathit{comp}\ (\mathit{Const}\ 3) = [\mathit{IConst}\ 3]\ \wedge$
$\mathit{comp}\ (\mathit{Var}\ 2) = [\mathit{ILoad}\ 2]\ \wedge$
$\mathit{comp}\ (\mathit{App}\ (\mathit{op}\ +)\ (\mathit{Const}\ 3)\ (\mathit{Var}\ 2)) = [\mathit{ILoad}\ 2,\ \mathit{IConst}\ 3,\ \mathit{IApp}\ (\mathit{op}\ +)]$
by *simp*

6.4 Corrección del compilador

Para demostrar que el compilador es correcto, probamos que el resultado de compilar una expresión y a continuación ejecutarla es lo mismo que interpretarla; es decir,

theorem $\mathit{ejec}\ (\mathit{comp}\ e)\ \mathit{ent}\ [] = [\mathit{valor}\ e\ \mathit{ent}]$
oops

El teorema anterior no puede demostrarse por inducción en *e*. Para demostrarlo por inducción, lo generalizamos a

theorem $\forall\ vs.\ \mathit{ejec}\ (\mathit{comp}\ e)\ \mathit{ent}\ vs = (\mathit{valor}\ e\ \mathit{ent})\#vs$
oops

En la demostración del teorema anterior usaremos el siguiente lema.

lemma *ejec-append*:
$\forall\ vs.\ \mathit{ejec}\ (xs@ys)\ \mathit{ent}\ vs = \mathit{ejec}\ ys\ \mathit{ent}\ (\mathit{ejec}\ xs\ \mathit{ent}\ vs)$ (**is** *?P xs*)
proof (*induct xs*)
show *?P* [] **by** *simp*
next

```
 fix a xs
 assume HI: ?P xs
 thus ?P (a#xs)
 proof (cases a)
  case IConst thus ?thesis using HI by simp
 next
  case ILoad thus ?thesis using HI by simp
 next
  case IApp thus ?thesis using HI by simp
 qed
qed
```

Una demostración más detallada del lema es la siguiente:

```
lemma ejec-append-2:
 ∀ vs. ejec (xs@ys) ent vs = ejec ys ent (ejec xs ent vs) (is ?P xs)
proof (induct xs)
 show ?P [] by simp
next
 fix a xs
 assume HI: ?P xs
 thus ?P (a#xs)
 proof (cases a)
  fix v assume C1: a=IConst v
  show ∀vs. ejec ((a#xs)@ys) ent vs = ejec ys ent (ejec (a#xs) ent vs)
  proof
   fix vs
   have ejec ((a#xs)@ys) ent vs = ejec (((IConst v)#xs)@ys) ent vs
    using C1 by simp
   also have ... = ejec (xs@ys) ent (v#vs) by simp
   also have ... = ejec ys ent (ejec xs ent (v#vs)) using HI by simp
   also have ... = ejec ys ent (ejec ((IConst v)#xs) ent vs) by simp
   also have ... = ejec ys ent (ejec (a#xs) ent vs) using C1 by simp
   finally show ejec ((a#xs)@ys) ent vs = ejec ys ent (ejec (a#xs) ent vs) .
  qed
 next
  fix n assume C2: a=ILoad n
  show ∀vs. ejec ((a#xs)@ys) ent vs = ejec ys ent (ejec (a#xs) ent vs)
  proof
   fix vs
   have ejec ((a#xs)@ys) ent vs = ejec (((ILoad n)#xs)@ys) ent vs
    using C2 by simp
```

also have ... $=$ *ejec* (*xs@ys*) *ent* ((*ent n*)#*vs*) **by** *simp*
also have ... $=$ *ejec ys ent* (*ejec xs ent* ((*ent n*)#*vs*)) **using** *HI* **by** *simp*
also have ... $=$ *ejec ys ent* (*ejec* ((*ILoad n*)#*xs*) *ent vs*) **by** *simp*
also have ... $=$ *ejec ys ent* (*ejec* (*a*#*xs*) *ent vs*) **using** *C2* **by** *simp*
finally show *ejec* ((*a*#*xs*)@*ys*) *ent vs* $=$ *ejec ys ent* (*ejec* (*a*#*xs*) *ent vs*) **.**
qed
next
fix *f* **assume** *C3*: *a*=*IApp f*
show $\forall$ *vs*. *ejec* ((*a*#*xs*)@*ys*) *ent vs* $=$ *ejec ys ent* (*ejec* (*a*#*xs*) *ent vs*)
proof
fix *vs*
have *ejec* ((*a*#*xs*)@*ys*) *ent vs* $=$ *ejec* (((*IApp f*)#*xs*)@*ys*) *ent vs*
using *C3* **by** *simp*
also have ... $=$ *ejec* (*xs@ys*) *ent* ((*f* (*hd vs*) (*hd* (*tl vs*)))#(*tl*(*tl vs*)))
by *simp*
also have ... $=$ *ejec ys ent* (*ejec xs ent* ((*f* (*hd vs*) (*hd* (*tl vs*)))#(*tl*(*tl vs*))))
using *HI* **by** *simp*
also have ... $=$ *ejec ys ent* (*ejec* ((*IApp f*)#*xs*) *ent vs*) **by** *simp*
also have ... $=$ *ejec ys ent* (*ejec* (*a*#*xs*) *ent vs*) **using** *C3* **by** *simp*
finally show *ejec* ((*a*#*xs*)@*ys*) *ent vs* $=$ *ejec ys ent* (*ejec* (*a*#*xs*) *ent vs*) **.**
qed
qed
qed

La demostración del teorema es la siguiente

theorem $\forall$ *vs*. *ejec* (*comp e*) *ent vs* $=$ (*valor e ent*)#*vs*
proof (*induct e*)
fix *v*
show $\forall$ *vs*. *ejec* (*comp* (*Const v*)) *ent vs* $=$ (*valor* (*Const v*) *ent*)#*vs* **by** *simp*
next
fix *x*
show $\forall$ *vs*. *ejec* (*comp* (*Var x*)) *ent vs* $=$ (*valor* (*Var x*) *ent*) # *vs* **by** *simp*
next
fix *f e1 e2*
assume *HI1*: $\forall$ *vs*. *ejec* (*comp e1*) *ent vs* $=$ (*valor e1 ent*) # *vs*
and *HI2*: $\forall$ *vs*. *ejec* (*comp e2*) *ent vs* $=$ (*valor e2 ent*) # *vs*
show $\forall$ *vs*. *ejec* (*comp* (*App f e1 e2*)) *ent vs* $=$ (*valor* (*App f e1 e2*) *ent*) # *vs*
proof
fix *vs*
have *ejec* (*comp* (*App f e1 e2*)) *ent vs*
$=$ *ejec* ((*comp e2*) @ (*comp e1*) @ [*IApp f*]) *ent vs* **by** *simp*

also have ... $=$ *ejec* ((*comp e1*) @ [*IApp f*]) *ent* (*ejec* (*comp e2*) *ent vs*)
 using *ejec-append* **by** *blast*
also have ... $=$ *ejec* [*IApp f*] *ent* (*ejec* (*comp e1*) *ent* (*ejec* (*comp e2*) *ent vs*))
 using *ejec-append* **by** *blast*
also have ... $=$ *ejec* [*IApp f*] *ent* (*ejec* (*comp e1*) *ent* ((*valor e2 ent*)#*vs*))
 using *HI2* **by** *simp*
also have ... $=$ *ejec* [*IApp f*] *ent* ((*valor e1 ent*)#((*valor e2 ent*)#*vs*))
 using *HI1* **by** *simp*
also have ... $=$ (*f* (*valor e1 ent*) (*valor e2 ent*))#*vs* **by** *simp*
also have ... $=$ (*valor* (*App f e1 e2*) *ent*) # *vs* **by** *simp*
finally
show *ejec* (*comp* (*App f e1 e2*)) *ent vs* $=$ (*valor* (*App f e1 e2*) *ent*) # *vs*
 by *blast*
qed
qed