Máquinas Virtuales

Compilación a Bytecode

28 de septiembre de 2023

CEK... eficiente?

La máquina CEK es un evaluador “eficiente” de expresiones FD4. pero todavía bastante alejada de un procesador real.

CEK... eficiente?

La máquina CEK es un evaluador "eficiente" de expresiones FD4. pero todavía bastante alejada de un procesador real.

- Analiza sintaxis abstracta (árboles): fácil desde Haskell... ¿pero en assembly? Hay que serializar el árbol.
- Completamente ligada al lenguaje: no podemos reusarla.
- Dos etapas, matching en elementos de la pila.

CEK... eficiente?

La máquina CEK es un evaluador "eficiente" de expresiones FD4. pero todavía bastante alejada de un procesador real.

- Analiza sintaxis abstracta (árboles): fácil desde Haskell... ¿pero en assembly? Hay que serializar el árbol.
- Completamente ligada al lenguaje: no podemos reusarla.
- Dos etapas, matching en elementos de la pila.

Buscamos una forma de evaluación más **directa**,
una secuencia de **instrucciones**.

```
(fun f -> 1 + f 3 4)
```

```
(fun f -> 1 + f 3 4)
```

```
movq $3,%rdi
movq $4,%rsi
call f
addq $1,%rax
```

```
(fun f -> 1 + f 3 4)
```

```
movq $3,%rdi
movq $4,%rsi
call f
addq $1,%rax
```

```
(fun f -> 1 + f 3 4)
```

```
movq $3, %rdi
movq $4, %rsi
call f
addq $1, %rax
```

Teorema Fundamental de la Ingeniería de Software

Teorema Fundamental de la Ingeniería de Software

Teorema

Todo problema en computación puede solucionarse agregando una capa de indirección.

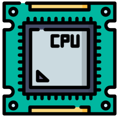
Teorema Fundamental de la Ingeniería de Software

Teorema

Todo problema en computación puede solucionarse agregando una capa de indirección.

(Salvo el problema de tener muchas indirecciones.)

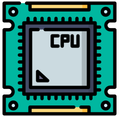

intel
CORE i7
X-series
AMD
RYZEN



Máquina Virtual

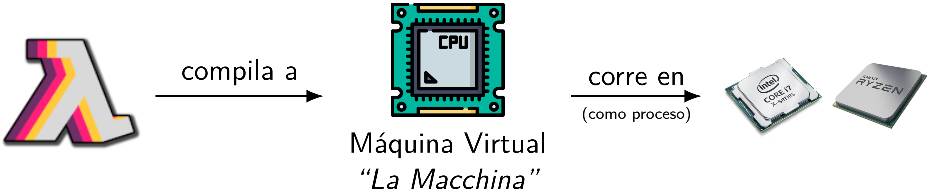
"La Macchina"

compila a



Máquina Virtual

"La Macchina"


compila a
CPU
Máquina Virtual
“La Macchina”
corre en
(como proceso)
intel
CORE i7
X-series
AMD
RYZEN

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**.

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El “hola mundo” de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) =$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \mathtt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \mathtt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \mathtt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) = \mathtt{CONST}(5);$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \mathtt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \mathtt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \mathtt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) = \mathtt{CONST}(5);\ \mathtt{CONST}(2);$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) = \texttt{CONST}(5);\ \texttt{CONST}(2);\ \texttt{NEG};$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El “hola mundo” de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) = \texttt{CONST}(5);\ \texttt{CONST}(2);\ \texttt{NEG};\ \texttt{CONST}(8);$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El “hola mundo” de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) = \texttt{CONST}(5);\ \texttt{CONST}(2);\ \texttt{NEG};\ \texttt{CONST}(8);\ \texttt{ADD};$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

$$\mathcal{C}(5 + ((-2) + 8)) = \texttt{CONST}(5);\ \texttt{CONST}(2);\ \texttt{NEG};\ \texttt{CONST}(8);\ \texttt{ADD};\ \texttt{ADD}$$

Máquinas de Pila

Las máquinas de pila ejecutan una secuencia de instrucciones, donde cada una tiene algún efecto en una **pila de valores**. El "hola mundo" de máquinas de pila: expresiones aritméticas.

$$e ::= N \mid e + e \mid -e$$

$$\begin{array}{rcl} \mathcal{C}(N) & = & \texttt{CONST}(N) \\ \mathcal{C}(e_1 + e_2) & = & \mathcal{C}(e_1);\ \mathcal{C}(e_2);\ \texttt{ADD} \\ \mathcal{C}(-e) & = & \mathcal{C}(e);\ \texttt{NEG} \end{array}$$

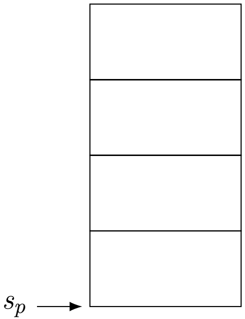
$$\mathcal{C}(5 + ((-2) + 8)) = \texttt{CONST}(5);\ \texttt{CONST}(2);\ \texttt{NEG};\ \texttt{CONST}(8);\ \texttt{ADD};\ \texttt{ADD}$$

Esencialmente, notación polaca inversa.

Máquinas de Pila — ejecución

`CONST(5);  CONST(2);  NEG;  CONST(8);  ADD;  ADD;  STOP`

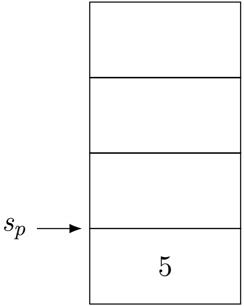
c_p



Máquinas de Pila — ejecución

`CONST(5); CONST(2); NEG; CONST(8); ADD; ADD; STOP`

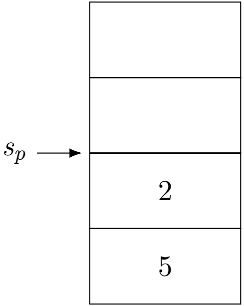
c_p



Máquinas de Pila — ejecución

CONST(5); CONST(2); NEG; CONST(8); ADD; ADD; STOP

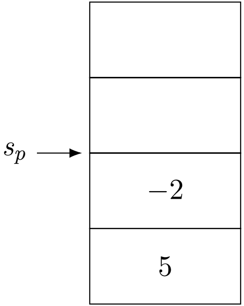
c_p



Máquinas de Pila — ejecución

CONST(5); CONST(2); NEG; CONST(8); ADD; ADD; STOP

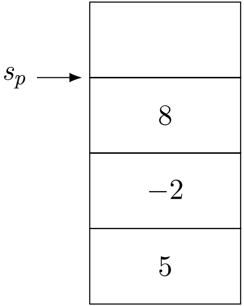
c_p



Máquinas de Pila — ejecución

$\texttt{CONST}(5);\quad \texttt{CONST}(2);\quad \texttt{NEG};\quad \texttt{CONST}(8);\quad \texttt{ADD};\quad \texttt{ADD};\ \texttt{STOP}$

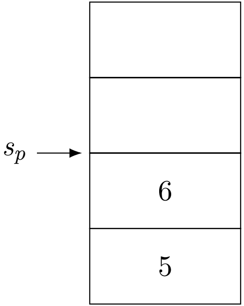
c_p



Máquinas de Pila — ejecución

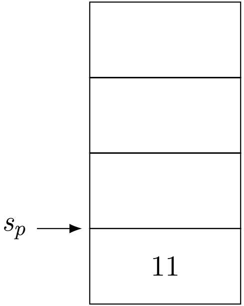
CONST(5); CONST(2); NEG; CONST(8); ADD; ADD; STOP

c_p



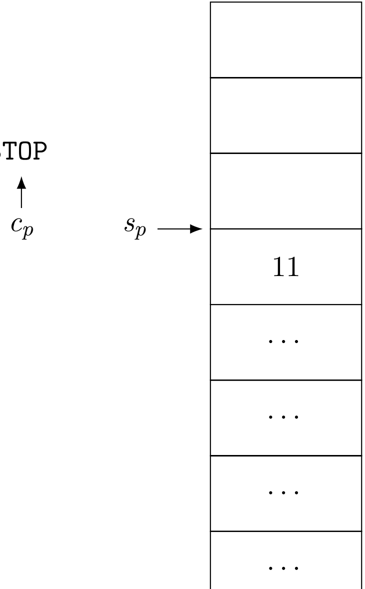
CONST(5); CONST(2); NEG; CONST(8); ADD; ADD; STOP

c_p



Máquinas de Pila — ejecución

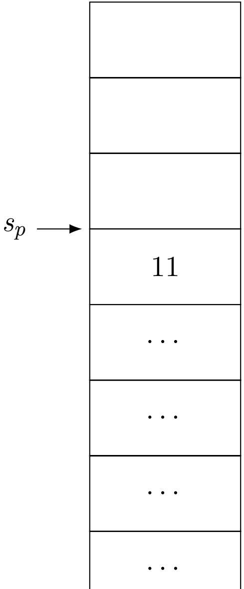
`CONST(5);  CONST(2);  NEG;  CONST(8);  ADD;  ADD;  STOP`



Máquinas de Pila — ejecución

`CONST`(5); `CONST`(2); `NEG`; `CONST`(8); `ADD`; `ADD`; `STOP`

$\uparrow$ c_p



Vamos a hacer **exactamente lo mismo**
para el fragmento aritmético de FD4.

Compilando el λ-cálculo — variables

La Macchina va a llevar también un **entorno** para las variables locales, similarmente a la CEK. La forma de un estado es $\langle c \mid e \mid s \rangle$.

Compilando el λ-cálculo — variables

La Macchina va a llevar también un **entorno** para las variables locales, similarmente a la CEK. La forma de un estado es $\langle c \mid e \mid s \rangle$. Es importante que c **es un puntero a código read-only**.

Compilando el λ-cálculo — variables

La Macchina va a llevar también un **entorno** para las variables locales, similarmente a la CEK. La forma de un estado es $\langle c \mid e \mid s \rangle$. Es importante que c **es un puntero a código read-only**.

$$\mathcal{C}(v_i) = \mathtt{ACCESS}(i)$$

$$\langle \mathtt{ACCESS}(i);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid e!i : s \rangle$$

Compilando el λ-cálculo — funciones

Las funciones tienen *instrucciones de retorno*

$$\begin{aligned}
\mathcal{C}(\lambda t) &= \texttt{FUNCTION}(\mathcal{C}(t);\ \texttt{RETURN}) \\
\mathcal{C}(f\ e) &= \mathcal{C}(f);\ \mathcal{C}(e);\ \texttt{CALL}
\end{aligned}$$

Compilando el λ-cálculo — funciones

Las funciones tienen *instrucciones de retorno*

$$\begin{array}{rcl} \mathcal{C}(\lambda t) & = & \texttt{FUNCTION}(\mathcal{C}(t);\ \texttt{RETURN}) \\ \mathcal{C}(f\ e) & = & \mathcal{C}(f);\ \mathcal{C}(e);\ \texttt{CALL} \end{array}$$

las llamadas proveen las *direcciones de retorno*, usadas por `RETURN`.

$$\begin{array}{rrrcrrr} \langle \texttt{FUNCTION}(c_f); c \mid & e \mid & s\rangle & \longrightarrow & \langle c \mid & e \mid & (e, c_f) : s\rangle \\ \langle \texttt{CALL}; c \mid & e \mid & v : (e_f, c_f) : s\rangle & \longrightarrow & \langle c_f \mid & v : e_f \mid & (e, c)_{RA} : s\rangle \\ \langle \texttt{RETURN}; \_ \mid & \_ \mid & v : (e, c)_{RA} : s\rangle & \longrightarrow & \langle c \mid & e \mid & v : s\rangle \end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x + 1)\ 10) =$$

Ejemplo

$$\mathcal{C}((\lambda x.x + 1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Ejemplo

$$\mathcal{C}((\lambda x.x + 1)\ 10) = \mathtt{FUNCTION}(\mathtt{ACCESS}\ 0;\ \mathtt{CONST}\ 1;\ \mathtt{ADD};\ \mathtt{RETURN});\ \mathtt{CONST}\ 10;\ \mathtt{CALL}$$

Suponemos una continuación k...

$$\langle \mathtt{FUNCTION}(B);\ \mathtt{CONST}\ 10;\ \mathtt{CALL};\ k \mid \qquad e \mid \qquad\qquad\qquad s \rangle \quad \longrightarrow$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \mathtt{FUNCTION}(\mathtt{ACCESS}\ 0;\ \mathtt{CONST}\ 1;\ \mathtt{ADD};\ \mathtt{RETURN});\ \mathtt{CONST}\ 10;\ \mathtt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rccl}
\langle \mathtt{FUNCTION}(B);\ \mathtt{CONST}\ 10;\ \mathtt{CALL};\ k \mid & e \mid & s \rangle & \longrightarrow \\
\langle \mathtt{CONST}\ 10;\ \mathtt{CALL};\ k \mid & e \mid & (e, B) : s \rangle & \longrightarrow
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s\rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s\rangle & \longrightarrow
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x + 1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s \rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s \rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s \rangle & \longrightarrow \\
\langle B \mid & 10 : e \mid & (e, k)_{RA} : s \rangle & =
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s\rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s\rangle & \longrightarrow \\
\langle B \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & = \\
\langle \texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & \longrightarrow
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s\rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s\rangle & \longrightarrow \\
\langle B \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & = \\
\langle \texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 10 : (e, k)_{RA} : s\rangle & \longrightarrow
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s\rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s\rangle & \longrightarrow \\
\langle B \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & = \\
\langle \texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 10 : (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 1 : 10 : (e, k)_{RA} : s\rangle & \longrightarrow
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s\rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s\rangle & \longrightarrow \\
\langle B \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & = \\
\langle \texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 10 : (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 1 : 10 : (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{RETURN} \mid & 10 : e \mid & 11 : (e, k)_{RA} : s\rangle & \longrightarrow
\end{array}$$

Ejemplo

$$\mathcal{C}((\lambda x.x+1)\ 10) = \texttt{FUNCTION}(\texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN});\ \texttt{CONST}\ 10;\ \texttt{CALL}$$

Suponemos una continuación k...

$$\begin{array}{rrrl}
\langle \texttt{FUNCTION}(B);\ \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 10;\ \texttt{CALL};\ k \mid & e \mid & (e, B) : s\rangle & \longrightarrow \\
\langle \texttt{CALL};\ k \mid & e \mid & 10 : (e, B) : s\rangle & \longrightarrow \\
\langle B \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & = \\
\langle \texttt{ACCESS}\ 0;\ \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{CONST}\ 1;\ \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 10 : (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{ADD};\ \texttt{RETURN} \mid & 10 : e \mid & 1 : 10 : (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle \texttt{RETURN} \mid & 10 : e \mid & 11 : (e, k)_{RA} : s\rangle & \longrightarrow \\
\langle k \mid & e \mid & 11 : s\rangle &
\end{array}$$

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\text{fix}.e) = \texttt{FIXPOINT}(e;\ \texttt{RETURN})$$

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\textsf{fix}.e) = \texttt{FIXPOINT}(e;\ \texttt{RETURN})$$

$$\left\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \right\rangle \longrightarrow \left\langle c \mid e \mid (e_{\text{fix}}, f) : s \right\rangle$$

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\textsf{fix}.e) = \texttt{FIXPOINT}(e;\ \texttt{RETURN})$$

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$

Para algún e_{fix}...

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\mathsf{fix}.e) = \texttt{FIXPOINT}(e;\ \texttt{RETURN})$$

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$

Para algún e_{fix}...

$$e_{\text{fix}} = (e_?, \qquad\qquad\qquad f) : e$$

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\mathsf{fix}.e) = \mathtt{FIXPOINT}(e;\ \mathtt{RETURN})$$

$$\left\langle \mathtt{FIXPOINT}(f);\ c \mid e \mid s \right\rangle \longrightarrow \left\langle c \mid e \mid (e_{\text{fix}}, f) : s \right\rangle$$

Para algún e_{fix}...

$$\begin{aligned} e_{\text{fix}} &= (e_?, \qquad\qquad\qquad\qquad f) : e \\ &= ((e_?, \qquad\quad f) : e, f) : e \end{aligned}$$

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\mathsf{fix}.e) = \texttt{FIXPOINT}(e;\ \texttt{RETURN})$$

$$\big\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \big\rangle \longrightarrow \big\langle c \mid e \mid (e_{\text{fix}}, f) : s \big\rangle$$

Para algún e_{fix}...

$$\begin{aligned} e_{\text{fix}} &= (e_?, \qquad\qquad\qquad f) : e \\ &= ((e_?, \qquad\quad f) : e, f) : e \\ &= (((e_?, f) : e, f) : e, f) : e \end{aligned}$$

Muy lindo pero... ¿Y los puntos fijos?

NO queremos llevar otro tipo de clausura como valor, pero tampoco podemos saber a priori si una f es recursiva.
¿Podemos convertir una clausura de fixpoint en una clausura normal?

$$\mathcal{C}(\mathsf{fix}.e) = \mathtt{FIXPOINT}(e;\ \mathtt{RETURN})$$

$$\langle \mathtt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$

Para algún e_{fix}...

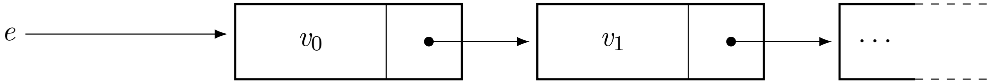
$$\begin{aligned} e_{\text{fix}} &= (e_?, \qquad\qquad\qquad f) : e \\ &= ((e_?, \qquad\quad f) : e, f) : e \\ &= (((e_?, f) : e, f) : e, f) : e \end{aligned}$$

¿Y si...?

$$e_{\text{fix}} = (e_{\text{fix}}, f) : e$$

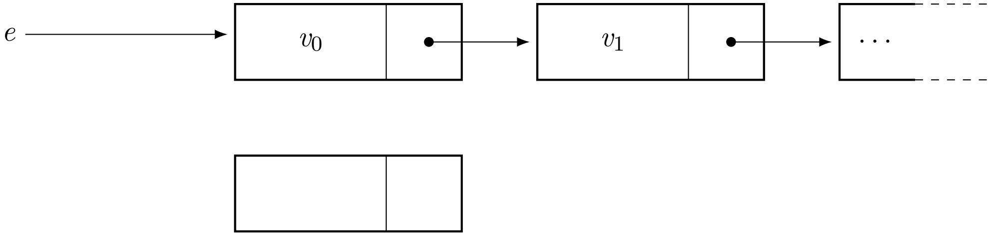
La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$



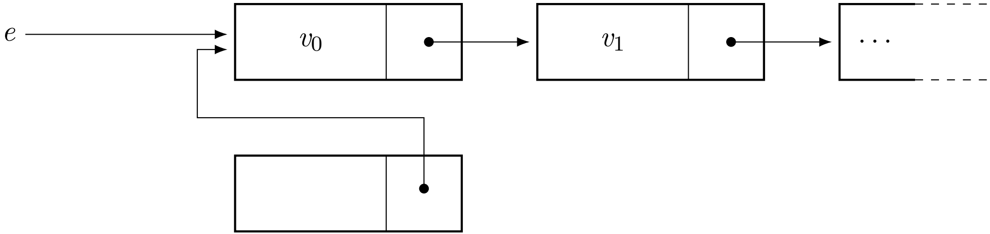
La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\mathrm{fix}}, f) : s \rangle$$



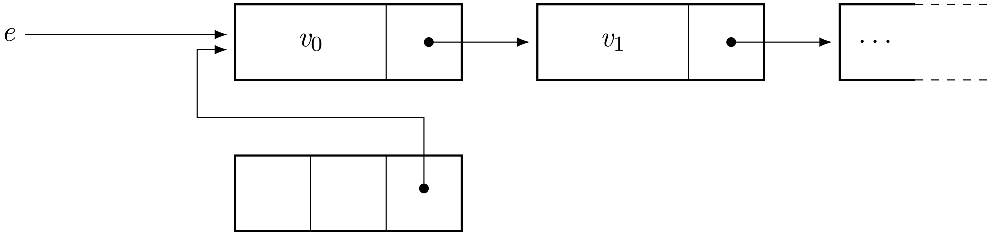
La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$



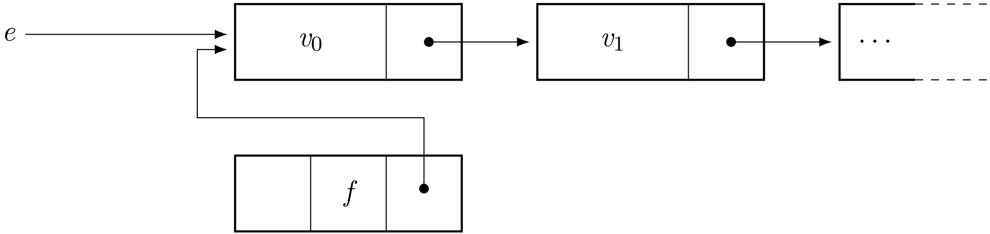
La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s\rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s\rangle$$



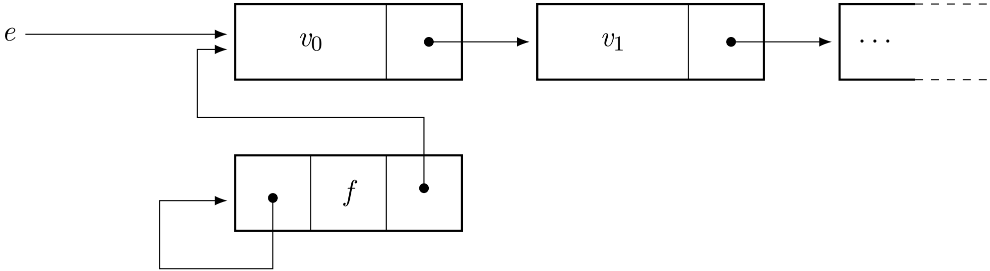
La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$



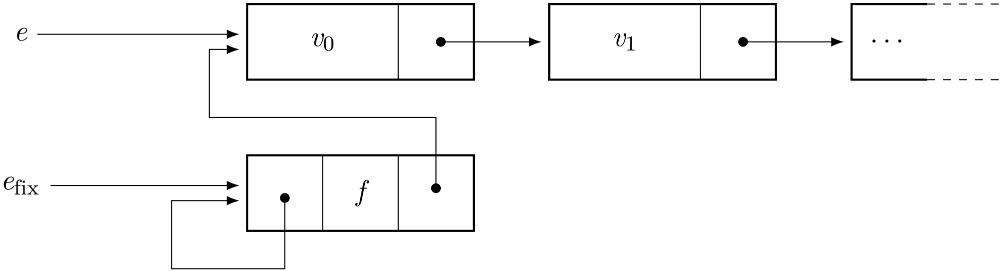
La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\mathrm{fix}}, f) : s \rangle$$



La verdad del punto fijo

$$\langle \texttt{FIXPOINT}(f);\ c \mid e \mid s \rangle \longrightarrow \langle c \mid e \mid (e_{\text{fix}}, f) : s \rangle$$



Es una *estructura cíclica* (como `let ones = 1:ones` en anabólicos).

Serializando de verdad

El bytecode tiene que ser una cadena de enteros, nada más. Para compilar los nodos `FUNCTION`, debemos dejar información para **saltar** el cuerpo.

Serializando de verdad

El bytecode tiene que ser una cadena de enteros, nada más. Para compilar los nodos `FUNCTION`, debemos dejar información para **saltar** el cuerpo.

$$\begin{array}{rcl} e & \rightsquigarrow & [10, 20, 30, 40] \\ \texttt{FUNCTION}(e) & \rightsquigarrow & [0x42, 4, 10, 20, 30, 40] \end{array}$$

Serializando de verdad

El bytecode tiene que ser una cadena de enteros, nada más. Para compilar los nodos `FUNCTION`, debemos dejar información para **saltar** el cuerpo.

$$\begin{aligned} e &\rightsquigarrow [10, 20, 30, 40] \\ \texttt{FUNCTION}(e) &\rightsquigarrow [0x42, 4, 10, 20, 30, 40] \end{aligned}$$

La máquina consume el "opcode", luego la longitud, y salta lo necesario hacia adelante.

Compilación de archivos

Esencialmente traducir:

$$
\begin{array}{l}
\texttt{let}\ v_1 = e_1 \\
\texttt{let}\ v_2 = e_2 \\
\ldots \\
\texttt{let}\ v_n = e_n
\end{array}
\longrightarrow
\begin{array}{l}
\texttt{let}\ v_1 = e_1\ \texttt{in} \\
\quad \texttt{let}\ v_2 = e_2\ \texttt{in} \\
\quad\quad \ldots \\
\quad\quad\quad \texttt{let}\ v_n = e_n\ \texttt{in} \\
\quad\quad\quad\quad 0
\end{array}
$$

y compilar ese término.

Resumen

- Una VM permite una compilación más simple.
- El bytecode puede ser portado fácilmente (e.g. Erlang)
- Las capacidades de la máquina se eligen a medida
- JIT: ver apunte.

Tareas:

- Implementar compilación y máquina en Haskell
- Hay un esqueleto de código en el repo