Universidad Simón Bolívar

Tarea de Investigación 1

José Pascarella 11-10743

19 de abril de 2015

1. Estrategias de Evaluación

1.1. Orden Aplicativo

```
(λo.K) ((λf.f f) (λf.λx.f [f x]) A H)
⟹ (λo.K) ((λf.λx.f [f x]) (λf.λx.f [f x]) A H)
⟹ (λo.K) ((λx.[λf.λi0.f (f i0)] [(λf.λi0.f (f i0)) x]) A H)
⟹ (λo.K) ((λf.λx.f [f x]) ([λf.λx.f (f x)] A) H)
⟹ (λo.K) ((λf.λx.f [f x]) (λx.A [A x]) H)
⟹ (λo.K) ((λx.[λi0.A (A i0)] [(λi0.A (A i0)) x]) H)
⟹ (λo.K) ((λx.A [A x]) ([λx.A (A x)] H))
⟹ (λo.K) ((λx.A [A x]) (A [A H]))
⟹ (λo.K) (A (A [A (A H)]))
⟹ K
```

1.2. Orden Normal

```
(λo.K) ((λf.f f) (λf.λx.f [f x]) A H)
⟹ K
```

1.3. Orden de Aplicación

Se puede notar con este ejemplo que el orden de reducción usado puede afectar en el resultado, usando el método aplicativo no terminaria nunca, dado que el término se sigue agrandando.

$(\lambda x\,.\,z)\ ((\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w))$
$\Longrightarrow\ (\lambda x\,.\,z)\ ((\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w))$
$\Longrightarrow\ (\lambda x\,.\,z)\ ((\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w))$
$\Longrightarrow\ (\lambda x\,.\,z)\ ((\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w))$
$\Longrightarrow\ \ldots$

En cambio usando el metodo normal se llega rápidamente a la forma normal de la expresión.

$(\lambda x\,.\,z)\ ((\lambda w.w\ w\ w)\ (\lambda w.w\ w\ w))$
$\Longrightarrow\ z$

1.4. Teorema Church-Rosser

No existe dicho término, dado que el teorema Church-Rosser asegura que para cualquier expresión del λ-cálculo, si existe una forma normal, es única.

1.5. Termino sin forma Normal

En el siguiente término se puede apreciar que no existe una forma normal posible, si seguimos aplicando β-reducciones obtendremos el mismo término que se tenia al comienzo.

$(\lambda f\,.\,f\ \ f)\ (\lambda f\,.\,f\ \ f)$
$\Longrightarrow\ (\lambda f\,.\,f\ \ f)\ (\lambda f\,.\,f\ \ f)$
$\Longrightarrow\ \ldots$
$\Longrightarrow\ (\lambda f\,.\,f\ \ f)\ (\lambda f\,.\,f\ \ f)$

2. Codificación de Church

2.1. Pares

2.1.1. Definición de swap

Se define la función swap que recibe un par y devuelve el par invertido.

$\mathsf{swap} \equiv \lambda t.(\mathsf{pair}\ (\mathsf{second}\ t)\ (\mathsf{first}\ t))$

2.1.2. Forma normal de swap

$$
\begin{aligned}
&\lambda t.\mathsf{pair}\ (\mathsf{second}\ t)\ (\mathsf{first}\ t)\\
\Longrightarrow\ &\lambda t.(\lambda b.\lambda c.c\ (\mathsf{second}\ t)\ b)\ (\mathsf{first}\ t)\\
\Longrightarrow\ &\lambda t.\lambda c.c\ (\mathsf{second}\ t)\ (\mathsf{first}\ t)\\
\Longrightarrow\ &\lambda t.\lambda c.c\ (t\ (\lambda a.\lambda b.b))\ (\mathsf{first}\ t)\\
\Longrightarrow\ &\lambda t.\lambda c.c\ (t\ (\lambda a.\lambda b.b))\ (t\ (\lambda a.\lambda b.a))
\end{aligned}
$$

$\mathsf{swap} \equiv \lambda t.\lambda c.c\ (t\ (\lambda a.\lambda b.b))\ (t\ (\lambda a.\lambda b.a))$

2.2. Lógica Proposicional

2.2.1. Definiciones de los operadores lógicos

$$
\begin{aligned}
&\mathsf{false} \equiv \lambda x.\lambda y.y\\
&\mathsf{true} \equiv \lambda x.\lambda y.x\\
&\mathsf{and} \equiv \lambda p.\lambda q.p\ q\ p\\
&\mathsf{not} \equiv \lambda p.\lambda a.\lambda b.p\ b\ a\\
&\mathsf{or} \equiv \lambda p.\lambda q.p\ p\ q
\end{aligned}
$$

2.2.2. Uso de los operadores propuestos

$$
\begin{aligned}
&\mathsf{or}\ \mathsf{False}\ (\mathsf{not}\ \mathsf{False})\\
\Longrightarrow\ &\mathsf{False}\ \mathsf{False}\ (\mathsf{not}\ \mathsf{False})\\
\Longrightarrow\ &(\lambda y.y)\ (\mathsf{not}\ \mathsf{False})\\
\Longrightarrow\ &\mathsf{not}\ \mathsf{False}\\
\Longrightarrow\ &\lambda a.\lambda b.\mathsf{False}\ b\ a\\
\Longrightarrow\ &\lambda a.\lambda b.(\lambda y.y)\ a\\
\Longrightarrow\ &\lambda a.\lambda b.a\\
\Longrightarrow\ &\mathsf{True}
\end{aligned}
$$

2.3. Aritmética

2.3.1. Definición de add

Función que toma como parámetro dos números en forma λ.

```
add ≡ λm.λn.λf.λx.m f (n f x)
```

2.3.2. Uso de add

Término equivalente al natural 3:

```
succ (succ (succ zero))
⟹ λf.λz.succ (succ zero) f (f z)
⟹ λf.λz.(λi0.λi1.succ zero i0 (i0 i1)) f (f z)
⟹ λf.λz.(λi0.succ zero f (f i0)) (f z)
⟹ λf.λz.succ zero f (f (f z))
⟹ λf.λz.(λi0.λi1.zero i0 (i0 i1)) f (f (f z))
⟹ λf.λz.(λi0.zero f (f i0)) (f (f z))
⟹ λf.λz.zero f (f (f [f z]))
⟹ λf.λz.(λi0.i0) (f (f [f z]))
⟹ λf.λz.f (f (f z))
```

Suma de 1 + 2:

```
add (succ zero) (succ (succ zero))
⟹ (λn.λf.λx.succ zero f (n f x)) (succ (succ zero))
⟹ λf.λx.succ zero f (succ (succ zero) f x)
⟹ λf.λx.(λi0.λz.zero i0 (i0 z)) f (succ (succ zero) f x)
⟹ λf.λx.(λz.zero f (f z)) (succ (succ zero) f x)
⟹ λf.λx.zero f (f (succ [succ zero] f x))
⟹ λf.λx.(λz.z) (f (succ [succ zero] f x))
⟹ λf.λx.f (succ (succ zero) f x)
⟹ λf.λx.f ((λi0.λz.succ zero i0 [i0 z]) f x)
⟹ λf.λx.f ((λz.succ zero f [f z]) x)
⟹ λf.λx.f (succ zero f (f x))
⟹ λf.λx.f ((λi0.λz.zero i0 [i0 z]) f (f x))
⟹ λf.λx.f ((λz.zero f [f z]) (f x))
⟹ λf.λx.f (zero f (f [f x]))
⟹ λf.λx.f ((λz.z) (f [f x]))
⟹ λf.λx.f (f (f x))
```

α-equivalente al número 3.

2.4. Listas

Definición de la función length que recibe una lista y devuelve el número de elementos que posee.

```
length ≡ λL.L (λh.succ) nil
```

```
length ≡ λL.L (λh.λn.λf.λz.n f (f z)) (λf.λz.z)
```