Lenguajes de Programación
Auxiliar No3

Auxiliares: Kenji Maillard

27/03/2020

1 Resumen

- Lenguage con enlazadores y expresión aritmética (WAE)

  ⟨*expr*⟩ ::= `{ num` ⟨*val*⟩ `}`
  | `{` ⟨*binop*⟩ ⟨*expr*⟩ ⟨*expr*⟩ `}`
  | `{ idx` ⟨*id*⟩ `}`
  | `{ with { ` ⟨*id*⟩ ⟨*expr*⟩ `} ` ⟨*expr*⟩ `}`

  Todos los lenguages en los ejercicios son extensiones de WAE.

Si tiene una duda sobre una función, puedes usar el Help Desk (menú Help en DrRacket). **Recuerde enunciar el contrato y escribir tests.**

2 Ejercicios

1. *Efectos y orden de evaluación*

   ⟨*expr*⟩ ::= ...
   | `{ idx` ⟨*id*⟩ `}`
   | `{ with-eager { ` ⟨*id*⟩ ⟨*expr*⟩ `} ` ⟨*expr*⟩ `}`
   | `{ with-lazy { ` ⟨*id*⟩ ⟨*expr*⟩ `} ` ⟨*expr*⟩ `}`

   (a) Defina una función `eval` evaluando una expresión, utilizando evaluación temprana para `with-eager` y perezosa para `with-lazy`.

   (b) Añada una operación `{ print` ⟨*expr*⟩ `}` para imprimir el resulto de una expresión antes de retornarlo.

(c) De ejemplos de expresiones donde el uso de `with-eager` o `with-lazy` cambia lo que se puede observar.

2. Defina la función `(free-vars expr)` que retorna la lista de variables libres en la expresión WAE `expr`.

3. *Locally nameless* La tecnica de locally nameless emplea índices de De Bruijn para identificadores enlazados y nombres (o symbolos) para identificadores libres:

   ⟨*ln-expr*⟩ ::= ...
   | `{ bound-id` ⟨*num*⟩ `}`
   | `{ free-id` ⟨*id*⟩ `}`
   | `{ with-ln` ⟨*ln-expr*⟩ ⟨*ln-expr*⟩ `}`

   (a) Defina una función que convierte una `expr` WAE a una `ln-expr`.

   (b) Defina substitución y un intérprete para `ln-expr`.

4. *Ofuscación*. Implemente la función `obfuscate` que cambia los identificadores de un programa WAE a nombres aleatorios (use `gensym` para generar los nombres). La transformación debe respetar el alcance léxico. Ejemplo:

```
> (obfuscate (parse '{with {x 1} x}))
(with 'g108 (num 1) (id 'g108))

> (obfuscate (parse '{with {x 1} {with {y x} {+ x y}}}))
(with 'g109 (num 1) (with 'g110 (id 'g109) (add (id 'g109) (id 'g110))))
```

5. Consideramos el lenguage WAE extendido con función que toman un argumento.

   ⟨*expr*⟩ ::= ...
   | `{ app` ⟨*fun-name*⟩ ⟨*expr*⟩ `}`
   ⟨*FunDef*⟩ ::= `( fundef` ⟨*fun-name*⟩ ⟨*id*⟩ ⟨*expr*⟩ `)`

   Recordamos que la función de evaluación ahora tiene la firma siguiente:

```
interp : expr x list FunDef -> num
```

   (a) Añadir un test `{ ifz` ⟨*expr*⟩ ⟨*expr*⟩ ⟨*expr*⟩ `}`

   (b) Defina funciones `sum` y `fib` en WAE que calculan la suma de los numeros hasta el argumento y los numeros de fibonacci. Recuerde que:

   - fib 0 = fib 1= 1
   - fib (n + 2) = fib (n+1) + fib n

   (c) Defina las funciones mutualmente recursivas `even?` y `odd?` en WAE.