Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

Tarea 03

Karla Ramírez Pulido karla@ciencias.unam.mx
J. Ricardo Rodríguez Abreu ricardo_rodab@ciencias.unam.mx
Manuel Soto Romero manu@ciencias.unam.mx

Fecha de inicio: 02 de abril del 2018
Fecha de entrega: 11 de abril
Semestre 2018-2

1. Responder correctamente cada inciso

1.1 (2 ptos) Leer cuidadosamente y resolver el ejercicio 1.1.1

Definición 1.1: Un lenguaje `Turing Completo` es aquel que puede simular todas las acciones que una `máquina de Turing` realiza ante un cómputo, es decir, puede emular a una `Máquina de Turing`[1].

Para probar que un lenguaje es `Turing Completo`, se debe demostrar que podemos escribir todos los cómputos que una `Máquina de Turing` puede realizar en el lenguaje.[1]

1.1.1 En clase se ha expuesto el lenguaje `FWAE` que es un lenguaje con expresiones arirtméticas, funciones y aplicaciones de funciones. ¿Es `FWAE` un lenguaje `Turing Completo`? Se deberá proveer una respuesta breve e inambigua, seguida de una justificación más extensa de la respuesta.[2]

1.2 (3 ptos) Investigar si los siguientes lenguajes de programación hacen uso de la evaluación glotona o perezosa:

- Shell
- Python
- Ruby

Escribir un pequeño script para determinar la respuesta a cada pregunta.

[1] Se recomienda ver los enlace de ayuda de la sección tres.
[2] Hint: Usar el combinador Y.

1.3 (2 ptos) Realizar los siguientes ejercicios de investigación:

- ¿Las funciones `delay` y `force` de Scheme/Racket hacen que estos lenguajes sean perezosos?
- ¿Cómo simulan la evaluación perezosa si el régimen de evaluación es glotona?

Ejemplo de uso de `delay` y `force` en Racket.

```
;; Ejemploi del código:
(define obj (delay (+ 2 3)))
obj
(force obj)
obj
```

1.4 (5 ptos) Leer cuidadosamente y resolver los ejercicios 1.4.1, 1.4.2 y 1.4.3.

Definición 4.1: Se dice que una expresión E en un lenguaje L es reducible a otra expresión E' en el lenguaje L' si $\exists$ f función, tal que que $f(E') = E$ y se representa como $E \leq_f E'$. [3]

Definición 4.2: Se define $OUTPUT$ como la función que recibe una expresión E de un programa y regresa una respresentación única $R = OUTPUT(E)$ tal que $R \in \Gamma^*$ con Γ definido como el conjunto de símbolos que se representan en las cintas de una Máquina de Turing.[3][3] Ejemplo:[4]

```
def factorial(n):
    if n <= 0:
      return 1
    else:
      return n*factorial(n-1)

return = factorial(3)
```

Cuadro 1.4.1

```
def factorial(n)
    if n <= 0
     return 1
    else
      n*factorial(n-1)
    end
end

result = factorial(3)
```

Cuadro 1.4.2

Definición 4.3: Se dice que una expresión E en un lenguaje L es equivalente a otra expresión E' en un lenguaje L' si y sólo si $E \leq_f E'$ y $OUTPUT(E) = OUTPUT(E')$ y se denotará como $E \equiv E'$.

Demostrar o dar un contraejemplo para las siguientes proposiciones:

1.4.1 Demostrar que las expresiones 3.4.1 y 3.4.2 que se encuentran en el anexo son expresiones equivalentes. Suponer que el lenguaje `Gram` que se encuentra en el cuadro 3.2.1 es un lenguaje que implementa evaluación perezosa. Recordar que hasta ahora `FWAE` implementa evaluación glotona.

[3] $0, 1$para las computadoras.

[4] OUTPUT es equivalente a evaluar de forma binaria a la función interp ó calc en el lenguaje FWAE.

1.4.2 Sea E' una expresión en un lenguaje L' Turing Completo que implementa evaluación glotona. Demostrar que existe una expresión E'' en un lenguaje L'' Turing Completo que implementa evaluación perezosa tal que E' es equivalente a E''.

1.4.3 El resultado de una expresión ¿se ve afectado por el tipo de evaluación que el lenguaje posee? Se deberá justificar la respuesta.

2. Anexo

```
<num> ::= ... | −2 | −1 | 0 | 1 | 2 | ...
<bool> ::= true | false
<Gram> : = <num>
         | <bool>
         | var <id> = <Gram>
         | (<Gram> <op> <Gram>)
         | if <Gram> then <Gram> else <Gram> fi
         | def <id> (<id >∗) { <Gram>;∗ return <id >; }
         | <Gram> ; <Gram>
         | <Gram> ;
```

Gramática 3.3.1 (Lenguaje `Gram`)

```
var i = 0;
var j = ((i + 2) + 1);
var k = (j + 2);
def fun (x y) {
    var new = (x + y);
    return new;
}
var m = (fun(k j) + k);
var resultado = m;
resultado
```

Programa 3.3.1 (Lenguaje `Gram`)

```
;; Programa en FWAE
{with {i 0}
  {with {j {+ {+ i 2} 1}}
    {with {fun {fun{x y} {with {new {+ x y}} new}}
      {with {m {+ {fun {k j}} k}}
        {with {resultado m} resultado}}}}}}
```

Programa 3.4.2 (Lenguaje `FWAE`)

3. Sitio oficial de enlaces de ayuda

1. Sitio oficial de $\mu-$recursividad https://en.wikipedia.org/wiki/ %CE %9C-recursive_function Consultado el día:[21 de marzo de 2018]

2. ¿Cómo probar que un lenguaje es Turing Completo? https://www.i-ciencias.com/pregunta/5859/como-probar-un-lenguaje-de-programacion-es-TuringCompleto Consultado el día:[21 de marzo de 2018]

3. Definición de Turing Completo

https://www.cs.odu.edu/~zeil/cs390/s18/Public/turing-complete/index.html Consulta- do el día:[21 de marzo de 2018]

4. "Are there minimum criteria for a programming language being Turing complete?"

https://cs.stackexchange.com/questions/991/are-there-minimum-criteria-for-a-programming-language-being-turing-complete Consultado el día:[21 de marzo de 2018]

4. Bibliografía

[1] Shriram Krishnamurthi, David Tucker, Programming Languages: Application and Interpretation, Primavera-2017 Universidad de Brown, http://cs.brown.edu/courses/cs173/2012/book/book.pdf .

[2] Gerald Jay Sussman y Guy L. Steele, Jr. Scheme: An Interpreter for Extended Lambda Calculus, https://en.wikisource.org/wiki/Scheme:_An_Interpreter_for_Extended_Lambda_Calculus/Whole_text