Problema I:

En teoría y laboratorio hemos visto el lenguaje FAE, que es un lenguaje con expresiones aritméticas, funciones y aplicaciones de funciones. ¿Es FAE un lenguaje Turing-Completo?. Debes proveer una respuesta breve e inambigua, seguida de una justificación más extensa de tu respuesta. Hint: Investiguen sobre el combinador Y.

Definición Turing-Completo: El lenguaje Turing-Completo se define como sigue:

- Datos: $D_L = \{0, 1, \downarrow\}^*$
- Programas: P_L se definen mediante la siguiente gramática $P_L \ni p ::= 1 : \mathcal{I}_1; \ldots; m: \mathcal{I}_m : m + 1 : halt$

  $\mathcal{I} ::=$ right | left | write $\mathcal{S}$ | if $\mathcal{S}$ then goto ℓ else goto ℓ' | goto ℓ donde $\mathcal{S} \in \{0, 1, \downarrow\}$ y $\ell, \ell' \in N$.

- La función semántica $[\![\cdot]\!]$ se define mediante la siguiente semántica operacional:
  •Memorias: $S =_{def} D_L \times \{0, 1, \downarrow\} \times D_L$, donde usualmente escribiremos $LsR \in S$ en vez de $(L, s, R) \in S$. Obsérvese que una memoria puede verse como la cinta de una máquina de Turing.
  • Estados: $\varepsilon = N \times S$
  • Relación de transición: sea $p = 1 : \mathcal{I}_1; \ldots; m: \mathcal{I}_m : m + 1 : halt$
    - Si $\mathcal{I}_\ell$ = right *entonces* $p \rhd (\ell, Lss' R) \rightarrow (\ell + 1, Lss' R)$
    - Si $\mathcal{I}_\ell$ = right *entonces* $p \rhd (\ell, Ls) \rightarrow (\ell + 1, Ls\downarrow)$
    - Si $\mathcal{I}_\ell$ = left *entonces* $p \rhd (\ell, Lss' R) \rightarrow (\ell + 1, Lss' R)$
    - Si $\mathcal{I}_\ell$ = left *entonces* $p \rhd (\ell, sR) \rightarrow (\ell + 1, \downarrow sR)$
    - Si $\mathcal{I}_\ell$ = write $\mathcal{S}$ *entonces* $p \rhd (\ell, Ls'R) \rightarrow (\ell + 1, LsR)$
    - Si $\mathcal{I}_\ell$ = goto ℓ' *entonces* $p \rhd (\ell, Ls'R) \rightarrow (\ell', LsR)$
    - Si $\mathcal{I}_\ell$ = if $\mathcal{S}$ then goto ℓ' else goto ℓ'' *entonces* $p > (\ell, LsR) \rightarrow (\ell', LsR)$
    - Si $\mathcal{I}_\ell$ = if $\mathcal{S}$ then goto ℓ' else goto ℓ'' *entonces* $p > (\ell, Ls'R) \rightarrow (\ell'', Ls'R)$
  • Estados finales: $final(p) = \{(m + 1, \sigma) | \sigma \in S\}$.
  • Función de inicialización: $init : P_L \times D_L \rightarrow \varepsilon$, $init(x) = (1, \downarrow x)$
  • Función de salida: $output : P_L \times \varepsilon \rightarrow D_L$, $output ((\ell, LsR))$

Sí es Turing-Completo ya que cumple con las condiciones dadas por definición de Turing-Completo, el cual nos pide tener ciertas restricciones tales como variables, que aparecen en FAE marcados con id; condicionales, es decir, las posibles ramas que tiene nuestro lenguaje, en otras palabras, dada una expresión derivarla en saber si es un número (num), una suma (add), un identificador (id), ser una función (fun) o una aplicación de función (app).

Además por definición de FAE y la definición Turing-Completo se puede observar que la semántica operacional Turing-Completo simula al proceso de ejecución de una Máquina de Turing de una manera muy directa. También podemos decir que un programa (escrito en este lenguaje) es esencialmente la definición de la función de transición δ. Se intuye entonces que todo programa p puede transformarse en una máquina de Turing.

Sabemos que ser Turing completo implica saltos condicionales, los cuales los cumple FAE (que ya hemos explicado anteriormente) así como tener variables que se podrán manipular en el mismo. Incluso podríamos decir que por la estructura de FAE utiliza cálculo lambda, que fue diseñado con el objetivo de utilizar funciones, la aplicación de éstas, así como funciones recursivas (combinador Y).

Por definición de cálculo lambda este resulta ser universal porque cualquier función computable puede ser expresada y evaluada a través de él, lo cual nos lleva a decir que es parecido a las máquinas de Turing.

Problema II:

¿Java es glotón o perezoso? Escribe un programa para determinar la respuesta a esta pregunta. El mismo programa, ejecutado en cada uno de los dos regímenes, debe producir resultados distintos. Puedes usar todas las características de Java que gustes, pero debes mantener el programa relativamente corto: penalizaremos cualquier programa que consideremos excesivamente largo o confuso (Hint: es posible resolver este problema con un programa de unas cuantas docenas de líneas). Debes anexar tanto el código fuente de tu programa (en un archivo aparte al PDF de la tarea) así como una respuesta a la pregunta de si Java es glotón o perezoso, y una explicación de porque su programa determina esto.
Es decir, deben proveer una respuesta breve e inambigua (p.ej. ”Java es perezoso”) seguida de una descripción del resultado que obtendrías bajo cada régimen, junto con una breve explicación de por qué ese régimen generaría tal resultado.

Java es glotón ya que en cuanto le es posible evaluar, realiza la operación correspondiente y de esta manera realiza la asignación a la variable. Es por eso que nuestro programa lanza un error (NaN) cuando se intenta hacer evaluación perezosa, porque se está intentando asignar a una variable el valor de una expresión y para poder darle valor a esa variable primero requiere realizar la evaluación de la función (“perezosa”). En nuestro ejemplos, hasta que encuentra la expresión que manda a llamar a perezosa, obtiene el número en ese método y le hace logaritmo de lo que haya recibido como parámetro.
Bajo la evaluación glotona (que es la que ocupa Java) obtendremos un resultado inmediato de la resta. Bajo la evaluación perezosa va a cargar con la expresión hasta que ya tenga que devolver un resultado, lo que forzará primero a realizar la evaluación de la función y después ya podemos obtener un resultado.

Bibliografía:

- http://programmers.stackexchange.com/questions/132385/what-makes-a-language-turing-complete
- https://es.wikipedia.org/wiki/C%C3%A1lculo_lambda
- http://lya.fciencias.unam.mx/favio/publ/esp/mmturingVE.pdf