Problema 1

FAE es Turing-Completo.
Para que un lenguaje funcional pueda ser Turing-completo, necesita:

- Condicionales.
- Recursión.

FAE tiene funciones, aplicación de funciones, expresiones aritméticas y es perfectamente capaz de usar recursión.
Las condicionales de FAE pueden definirse con el cálculo lambda que el lenguaje nos permite utilizar.
La recursión de FAE en la aplicación de funciones se puede definir mediante el combinador Y, el cual se puede aplicar puesto que nuestro lenguaje nos permite usar el cálculo lambda.
Con esto, tenemos que podemos utilizar funciones de primera clase, y esto significa que podemos pasar funciones como argumento a otras funciones, regresar funciones como resultado de evaluar otra función, y crear funciones conforme las vayamos utilizando.
El combinador Y se usa para la recursión anónima, ya que la función no se manda a llamar explícitamente con su mismo nombre, ésta toma un argumento, el cual es una función no recursiva y regresa una función recursiva.

Problema 2

Por lo que estuve buscando en los interwebz, java es Smart/Lazy.
Pero debido al programa que hice (el cuál está anexado en la carpeta de la tarea con el nombre de "proof.java"), es una especie de glotón inteligente, ya que dependiendo del orden en el que pusiera los operadores.
Lo que hice es muy sencillo, creo un objeto con dos booleanos y en el main creo dos booleanos y opero con un OR lógico; el booleano del objeto (cuyo valor tiene que ser falso) es escrito primero, es decir [getA() || B], la llamada al método GET que devuelve el valor, sí es operada; el booleano cuyo fue creado en el main (y su valor tiene que ser true), es escrito primero, es decir [B || getA()], entonces lo que hace es que no evalúe getA() (lo que yo creo que es culpa del compilador), ya que lógicamente si valor es irrelevante, porque B es un true, y el OR se hace true con uno de los operandos siendo true.

Problema 4.

Tener una evaluación perezosa nos dice que una función no necesita ser evaluada, a menos que se requiera el resultado.
Sabemos que las llamadas a las funciones no son evaluadas necesariamente en el orden como aparecen en el código. Si las funciones modifican el estado, entonces nos importa el orden en el que son ejecutadas. Es por eso que los lenguajes con evaluación perezosa necesitan tener estados para poder verificar que las funciones se fueron ejecutando en el orden que queremos.
Si después de ver esto, permitimos las operaciones con estados, entonces regresamos al problema original, el cual es no saber realmente si las funciones se están ejecutando en el orden original o no. Es por eso que los lenguajes perezosos no aceptan operaciones con estados.