Tomás Maturana
19.186.973-7

Control Lenguajes de Programación

P3:

a) En estricto rigor, no sería necesario modificarle nada para que se pudiera hacer exactamente lo mismo que en un lenguaje que acepta más argumentos por función, pero sería algo incómodo escribir programas tan grandes, como (add 3 (add 4 (add 5 6))), en vez de (add 4 5 6). Pero se pueden hacer pequeñas modificaciones al lenguaje para que las funciones acepten más argumentos y que estén currificadas, de tal modo, se aceptarían cosas como (add 4 5 6) y por "debajo" el lenguaje seguiría haciendo (add 3 (add 4 (add 5 6))).

b) Son cosas distintas, ya que call-by-name implica que se evaluará un elemento cada vez que se requiera, pero si se evalúa con el ambiente en el que fue definido el elemento, se tendrá alcance estático, y si se evalúa con el ambiente con el que fue llamada la evaluación del elemento, se tendrá alcance dinámico.

c) No, ya que no se aceptaría mutación. Por otro lado, si es un lenguaje sin mutación, a una función se le puede pasar un argumento compuesto, es decir, que debe ser evaluado, entonces en la tabla tendrían que ir los input ya evaluados, luego, no tiene mucho sentido hacer eso ya que si se desea ocupar el valor retornado en algo posterior, basta con definir una variable que lo contenga. Además de esto, si bien podría en algunos casos optimizar el tiempo de respuesta del lenguaje, habría que hipotéticamente tener espacio ilimitado si se quiere correr muchas funciones, luego la eficiencia en espacio no sería para nada buena y más encima, si se tiene una tabla muy grande luego de correr varias funciones, la eficiencia en tiempo también podría ir empeorando.

d) Le diría "Khekhéee? D:", lazy evaluation implica evaluar los argumentos, o partes de una función sólo y cuando se va necesitar como un valor que necesito ocupar, entonces, mientras no lo necesite ocupar, el elemento estará "como recién salido del parser". La idea de Francisco se podría implementar tanto en lazy evaluation como en eager evaluation, con las mismas situaciones nefastas que provoca.

e) Tendrá la propiedad estática si no se necesita guardar el ambiente o variables que se generaron en la función misma, independizando el resultado, de la ejecución de esta función. Es decir, la función retorna un valor y termina (siendo ese valor o un valor como tal, o una función), liberando la memoria que se ocupó durante la ejecución de la función. Si tuviera una propiedad dinámica, la memoria no se liberaría, esperando ocupar eso en algún momento.
Si quisiéramos determinar las invocaciones por la cola de un programa, basta con ver el stack y verificar si luego de llamar la función, ésta retorna un valor o una llamada a otra función y luego termina, o si queda en suspenso hasta que su resultado tenga que retornar (en caso de retornar una función). En este último caso, se tendría una función que no es por

la cola.

f) Se debería agregar una fase entre el parsing y la evaluación. Debe pasar por el parsing, ya que éste verificará si la sintaxis de lo que se desea ejecutar es correcta. Luego, se preprocesa para ver si existen macros en lo que se desea ejecutar y así manipular los argumentos sin ser evaluados a menos que la macro lo requiera. Finalmente, se termina evaluando el resultado del preproceso.