Práctica de Laboratorio

Intérprete de Lambda Cálculo

Grado en Ingeniería Informática (Q7)
Diseño de Lenguajes de Programación
Curso 2021/22

1 Objetivos

Junto con el enunciado de esta práctica se proporcionan varias implementaciones de un intérprete de lambda cálculo escritas en OCaml, las cuales, por otra parte, están basadas en las implementaciones que Benjamin C. Pierce cita en su libro *Types and Programming Languages*, y ya han sido explicadas en las clases de prácticas. Dichas implementaciones procesan expresiones de lambda cálculo escritas ante el prompt de un lazo interactivo, las evalúan y escriben en pantalla los correspondientes resultados. El propósito de esta práctica es estudiar dichas implementaciones y ampliarlas en los términos que se explican a continuación. Se considerarán tanto mejoras en los propios intérpretes, como extensiones en el lenguaje lambda cálculo que reconocen.

Hemos visto tres versiones del intérprete: `lambda-1`, `lambda-2` y `lambda-3`. Las mejoras y extensiones que consideraremos no afectarán en ningún caso a las versiones `lambda-1` y `lambda-2`. En el primer caso, por ser demasiado básica y sobre todo por la incomodidad de manejo que implica el uso de los términos "Church booleans" y "Church numerals" para la representación de los valores lógicos y de los números naturales. Pero la razón principal, en ambos casos, es el hecho de no incluir lambda cálculo tipado. Así pues, afectarán únicamente a la versión más completa: `lambda-3`.

2 Apartados de la práctica

Las mejoras y extensiones propuestas en esta práctica pueden enumerarse bajo la forma de apartados, tal y como se indica a continuación:

1. Mejoras en la introducción y escritura de las lambda expresiones:

   1.1. Reconocimiento de expresiones multi-línea. Dado que en este caso el salto de línea por sí solo ya no implicará necesariamente el final de una expresión, quizás sea útil introducir algún nuevo símbolo o símbolos para indicar ese aspecto (por ejemplo, un doble punto y coma).

   Reconocimiento de líneas multi-expresión. Dado que ahora una línea podría contener no solo una expresión, sino varias, quizás sea útil introducir algún nuevo símbolo o símbolos para separar unas expresiones de otras dentro de la misma línea (por ejemplo, un punto y coma).

   Ambas mejoras pueden abordarse mediante la manipulación directa del *string* que teclea el usuario cuando introduce un nuevo término y/o mediante la introducción de

nuevos tokens en el analizador léxico y de nuevas reglas gramaticales en el analizador sintáctico (siendo esta última estrategia la más conveniente y elegante).

Este apartado es obligatorio.

1.2. Reconocimiento de expresiones desde fichero. El objetivo de este apartado no es simplemente el de poder evaluar expresiones que ya estén escritas en un fichero, en lugar de tener que teclearlas ante el prompt del intérprete. Eso ya podemos hacerlo con las implementaciones originales, simplemente redirigiendo la entrada estándar. Se espera, por tanto, algún comportamiento adicional. Por ejemplo, si el fichero contiene alguna expresión errónea, que el proceso de lectura y evaluación se detenga, indicando, además del tipo de error, el número de la línea donde se detectó dicho error.

Este apartado es opcional y aporta hasta 0,75 puntos.

1.3. Implementación de un "*pretty-printer*" más completo. El objetivo principal de este apartado es el de intentar minimizar el número de paréntesis a escribir cuando una expresión se transforma en un *string*. Quizás la mejor manera de hacerlo sea mediante una especie de cascada de funciones y subfunciones (guiadas por la gramática del lenguaje), que se van llamando unas a otras según se va profundizando en la estructura interna de la expresión. Existen otros aspectos adicionales que también podrían ser considerados, como por ejemplo la indentación de los diferentes elementos de las expresiones, pero la minimización de paréntesis es el aspecto esencial.

Este apartado es opcional y aporta hasta 1,5 puntos.

2. Mejoras en la evaluación del lambda-cálculo:

2.1. Incorporación de un "modo *debug*" que vaya escribiendo en pantalla los términos intermedios que va generando la función `eval1`, antes del término final que devuelve la función `eval`.

Este apartado es opcional y aporta hasta 0,25 puntos.

2.2. Estudio de los índices de De Bruijn, para la representación "*nameless*" de los términos, y re-implementación de la función de substitución tomando como base esta idea, lo cual puede requerir también la implementación de una operación auxiliar de desplazamiento ("*shifting*").

Este apartado es opcional y aporta hasta 1,5 puntos.

3. Ampliaciones del lenguaje lambda-cálculo:

3.1. Incorporación de un combinador de punto fijo interno, de tal forma que se puedan declarar este tipo de funciones mediante definiciones recursivas directas. La idea es que en lugar de escribir

```
let fix = lambda f.(lambda x. f (lambda y. x x y)) (lambda x. f (lambda y. x x y)) in
let sumaux =
   lambda f. (lambda n. (lambda m. if (iszero n) then m else succ (f (pred n) m))) in
let sum = fix sumaux in
sum 21 34
```

podamos escribir

```
letrec sum : Nat -> Nat -> Nat =
   lambda n : Nat. lambda m : Nat. if iszero n then m else succ (sum (pred n) m) in
sum 21 34
```

Con el fin de verificar más exhaustivamente el correcto comportamiento de esta nueva ampliación, escriba en el fichero `examples.txt` proporcionado y en la memoria final de la práctica dos ejemplos adicionales que involucren doble recursividad a partir de la suma: una lambda expresión que calcule el producto de dos números naturales y otra que calcule el término n-ésimo de la serie de Fibonacci.

Este apartado es obligatorio.

3.2. Incorporación de un contexto de variables, que permita asociar nombres de variables libres con valores o términos, de forma que éstos puedan ser utilizados en lambda expresiones posteriores. La sintaxis debe ser

```
identificador = término
```

Por ejemplo:

```
x = true
id = lambda x : Bool. x
```

De esta forma, si después escribimos

```
id x
```

esta expresión debe ser válida y debe devolver `true`.

Se recomienda reflexionar sobre cómo debe comportarse esta nueva funcionalidad si durante la definición de una nueva variable se reasigna un nuevo valor a un nombre que ya está en el contexto (es decir, se reutiliza un nombre de variable que ya se utilizó en una definición de variable previa). En función de cómo se maneje este fenómeno, podríamos estar ante un "contexto imperativo" o ante un "contexto funcional". Si estamos implementando un intérprete de lambda cálculo, lo más apropiado es que el contexto sea funcional. Y para ello existen diferentes alternativas de implementación. Se debe elegir una de ellas, explicando cómo se ha programado y explicando también la razón de dicha elección.

Este apartado es obligatorio.

3.3. Incorporación del tipo *string* para el soporte de cadenas de caracteres, así como de la operación de concatenación de estas cadenas.

Este apartado es obligatorio.

3.4. Incorporación de los pares (tuplas de dos elementos de cualquier tipo, incluso de tipos diferentes entre ellos), con las operaciones típicas de proyección *first* y *second*.

Este apartado es obligatorio.

3.5. Incorporación de los registros (secuencias finitas de campos de cualquier tipo etiquetados), con las operaciones típicas de proyección mediante la etiqueta del campo.

Este apartado es opcional y aporta hasta 1 punto.

3.6. Incorporación de las listas (secuencias finitas de elementos de un mismo tipo), con las operaciones típicas de cabeza, cola y es vacía. Adicionalmente, escriba en el fichero `examples.txt` proporcionado y en la memoria final de la práctica dos lambda expresiones: una que calcule recursivamente en función de la suma la longitud de una lista, y otra que realice la aplicación de una función sobre los elementos de una lista y devuelva la lista de los valores resultantes (*map*).

Este apartado es opcional y aporta hasta 1 punto.

3.7. Incorporación de subtipado. Más concretamente, se trataría de escribir una función que implemente el polimorfismo de subtipado para registros y funciones (es decir, una función que compruebe si dos tipos dados verifican dicha relación de subtipado), y de re-implementar la función general de tipado de forma que use este tipo de polimorfismo allí donde sea aplicable.

Este apartado es opcional y aporta hasta 2 puntos.

4. Redacción de una memoria final:

4.1. Además de entregar el código fuente, debe entregarse también una **memoria** en formato pdf que contenga lo siguiente: un pequeño **manual de usuario** que ilustre (formalmente y con ejemplos de ejecución) las nuevas funcionalidades del intérprete, y también un pequeño **manual técnico** que indique qué módulos de las implementaciones originales han sido modificados y qué tipo de cambios se han realizado sobre ellos para lograr implementar esas nuevas funcionalidades (si bien este segundo manual puede ser sustituido por comentarios en el propio código fuente, siempre y cuando éstos sean suficientemente claros y cumplan la misión anteriormente indicada).

ingles

Este apartado es obligatorio.

3 Instrucciones de entrega y evaluación

A continuación se enumeran las instrucciones de entrega y evaluación de esta práctica:

- La realización de los apartados obligatorios aporta hasta 5 puntos. Cuando decimos "hasta" nos referimos a que se podrá alcanzar esa cantidad de puntos siempre y cuando los apartados funcionen correctamente y sin anomalías. Si este aspectos no se cumple, se aplicarán penalizaciones. Esto mismo es aplicable a los apartados opcionales.

- De cara a la valoración de cada trabajo entregado, también se tendrá en cuenta, entre otros aspectos, la usabilidad del programa, así como la claridad y calidad (y no necesariamente la extensión) de los comentarios del código. Cabe aclarar, en lo que se refiere a los apartados opcionales, que su realización será valorada de acuerdo con las puntuaciones citadas en cada uno de ellos, aunque su realización no es estrictamente necesaria para aprobar la práctica (eso sí, con una nota máxima de 5 puntos, en caso de que no se realice ninguno).

- La **fecha límite de entrega** de la práctica es el viernes 10 de diciembre de 2021. Los trabajos presentados requerirán sus correspondientes **defensas** ante el profesor de prácticas durante las sesiones de los días 13 y 20, o bien 14 y 21, según el grupo, de diciembre de 2021.