Compiladores 2020-1
Facultad de Ciencias UNAM
Práctica 5: Inferencia de Tipos

Lourdes del Carmen González Huesca Alejandra Krystel Coloapa Díaz
Javier Enríquez Mendoza

Integrantes del equipo

Ángeles Martínez Ángela Janín	314201009
García Landa Valeria	314033008
Martínez Monroy Emily	314212391
Rebollar Pérez Ailyn	314322164

Desarrollo de los Ejercicios

Ejercicio 1
Para este ejercicio lo que hicimos fue definir un nuevo lenguaje que se llama L9, donde estamos considerando tanto a las lambdas como a las aplicaciones de función currificadas, entonces sólo tenemos un parámetro y un cuerpo, así que lo que hicimos para las lambdas fue tomar la misma idea que implementamos en la práctica anterior de una función auxiliar que la currificara que se llama currifica que recibe cuatro cosas: una lista de variables, una lista de tipos, un cuerpo y la palabra lambda para el constructor de éstas.

Sin embargo, para la aplicación de función decidimos hacer una función que se llama curry-app, donde definimos en cada llamada el parámetro y la función, pero como teníamos problemas con la asociación de los paréntesis, lo que hicimos fue aplicar la reversa tanto en los parámetros como para las funciones y asociar como requería el ejercicio.

A continuación el lenguaje L9:

```
(define-language L9
  (extends L8)
  (Expr (e body)
        (- (lambda ([x* t*] ...) body)
           (e0 e1 ...))
        (+ (lambda ([x t]) body)
           (e0 e1))))
```

Ejercicio 2
Para este ejercicio se definió un nuevo lenguaje que se llama L10 donde eliminamos el constructor cuote y agregamos uno nuevo llamado const, donde ahí volvemos a englobar a las constantes que son enteros, booleanos y char, sólo que ahora agregamos su tipo, así que el lenguaje se ve de la siguiente manera:

```
(define-language L10
  (extends L9)
  (Expr (e body)
        (- (cuote c))
        (+ (const t c))))
```

Ahora sí, para éste proceso lo que hicimos fue cazar el constructor de cutote y checar si era un entero, un booleano o un char y de acuerdo a eso, devolvíamos el tipo adecuado Int, Bool o Char.

Sección: Inferencia
Ejercicio 1
Para éste ejercicio no se definió un nuevo lenguaje, sin embargo lo que hicimos fue ir siguiendo las reglas de inferencia que estaban en el pdf, así que las hipótesis las vimos como condiciones que se debían seguir.

Asimismo, hicimos una función auxiliar que se llama *look-up* para buscar a una variable con su tipo en el contexto, ésta función la requerimos para el caso de una variable, entonces sólo recibe una variable y el contexto en el que se quiere buscar que en nuestro caso es una lista de pares.

Para los casos donde se nos pide unificar, usamos la función auxiliar *unify* que ya venía en la práctica mientras que para el constructor (list e*...) ocupamos de la función part para tener el tipo más particular y poder unificar.

El caso que se nos dificultó fue la aplicación de función pues e0 podría ser cualquier cosa sintácticamente pero necesitamos que su tipo sea de función o bien (T $\rightarrow$ R) aunque tuvimos la duda si el constructor primapp entraba en éste caso, así que decidimos suponer que sólo serían lambdas para no complicarnos tanto e hicimos una función auxiliar que se llama *esFuncion?* que verifica que lo que reciba sea de la forma (T $\rightarrow$ R).

Ejercicio 2
Para éste ejercicio tampoco se definió un lenguaje nuevo, pero lo que se hizo fue cazar los constructores, lambda, let, letrec y letfun donde tenemos las variables y tipos, entonces, lo que se hizo fue que en lugar de regresar el tipo, regresamos lo que infiere el algoritmo J con la expresión que se le va a asignar con el tipo, eso en cada uno ya que el algoritmo J ya sabes que hacer con la lista vacía o una lista con muchos parámetros, lo mismo para las funciones o lambdas.