Trabajo Practico Final
Análisis del Lenguaje de Programación
Sistema F

Ramiro Gatto

.../03/2025

1. Descripción del Proyecto

La idea principal del proyecto es la de implementar un EDSL sobre el Sistema F, el cual permita la evaluación de algunos términos del mismo. Para que el proyecto sea simple se eligieron un par de tipos bases para el evaluador, los cuales son:

- Empty
- Booleanos
- Naturales
- Funciones
- Listas (de cualquier tipo)

Además, para el evaluador también se definió lo siguiente:

- Chequeador de tipos.
- Pretty-printer.
- Parser.

Para poder realizar el mismo se tomo como inspiración el Trabajo Practico Nº2 [1], el cual se extendió/modifico para satisfacer con lo requerido.

2. Manual de uso e Instalación del software

Para poder usar el evaluador se va a necesita de Stack [2], una vez instalado se tiene que abrir una consola en el directorio del proyecto (Sistema-F) y ejecutar:

1. stack setup (una única vez)
2. stack build
3. stack exec Sistema-F-exe

Con las dos primeras lineas compilamos el proyecto y con la tercera lo ejecutamos.

3. Organización de los archivos

La organización de los archivos del proyecto es la siguiente:

```
Sistema-F
├── .stack-work
├── app
│   └── Main.hs
├── src
│   ├── Common.hs
│   ├── Parse.y
│   ├── PrettyPrinter.hs
│   └── SystamF.hs
├── .gitignore
├── CHANGELOG.md
├── Ejemplos.txt
├── LICENSE
├── package.yaml
├── README.md
├── Sistema-F.cabal
├── stack.yaml
└── stack.yaml.lock
```

Para entender como funciona el proyecto veamos la función de los archivos en las carpetas app y src que son los principales para el funcionamiento del mismo, el resto de los archivos son de configuración.

3.1. app

3.1.1. Main.hs

Este archivo es donde comienza la ejecución del programa al compilarse y ejecutarse (implementa el ejecutable final). Hace uso de una serie de funciones para poder parser la entrada por teclado, determinar el comando ingresado, imprimir por pantalla, realizar la evaluación, entre muchas cosas más.

Gran parte del **Main.hs** es idéntico al del TP Nº2 [1], salvo por algunas modificaciones y simplificaciones para que se ajuste al comportamiento requerido.

3.2. src

3.2.1. Common.hs

En este archivo es donde se encuentran las definición de los tipo y valores. Es decir es donde se definen los términos y valores que utilizara el Sistema F.

3.2.2. Parse.y

Para generar el parser utilizaremos happy. En este archivo se generan los parsers a utilizar, se definen los tokens que acepta el parse, entre más funciones. También es donde se define el lexer que se utilizara como analizador lexicográfico de la entrada.

Para crear el archivo se utilizo el Parser.y del TPº2 [1] y la documentación de Happy [3]. Se tiene 2 parses, uno es el parseStmt en donde el no terminal de nivel superior (top-level non-terminal) es **Def** y el segundo en donde el top-level non-terminal es **Exp**.

3.2.3. PrettyPrinter.hs

Para poder mostrar los términos se utilizo la biblioteca pretty printing, la cual consiste en una serie de combinados desarrollada por John Hughes. Aca es donde se encuentra implementado el pretty printer para el Sistema F.
Aca podemos distinguir 2 función importantes:

- pp: la cual dado un Term lo imprime por pantalla.
- printType: la cual se encarga de imprimir por pantalla los Type.

3.2.4. SystamF.hs

En este archivo es donde se implementan las funciones de evaluación y el chequeador de tipo (además de unas cuantas funciones auxiliares para el correcto funcionamiento).

4. Decisiones de diseño importantes

4.1. Representación del Sistema F

Los tipos, valores y términos en el Sistema F están dados por la siguientes gramática, respectivamente:

$$
\begin{aligned}
T &::= E \mid T \to T \mid X \mid \forall X\ .\ T \mid Bool \mid Nat \mid List\ T \\
v &::= True \mid False \mid nv \mid \lambda x : T.\ t \mid \Lambda X\ .\ t \\
nv &::= 0 \mid suc\ nv \\
t &::= x \mid \lambda x : T.\ t \mid t\ t \mid ifthenelse\ t\ t\ t \mid \Lambda X\ .\ t \mid t\ \langle X \rangle
\end{aligned}
$$

Como se menciono anteriormente, la implementación de estos se encuentra en el archivo **src/Common.hs** y es la siguiente:
Para los tipos es:

```
data Type = EmptyT
          | ListTEmpty
          | FunT Type Type
          | BoundForAll Int
          | VarT String
          | ForAllT Type
          | BoolT
          | NatT
          | ListT Type
          deriving (Show, Eq)
```

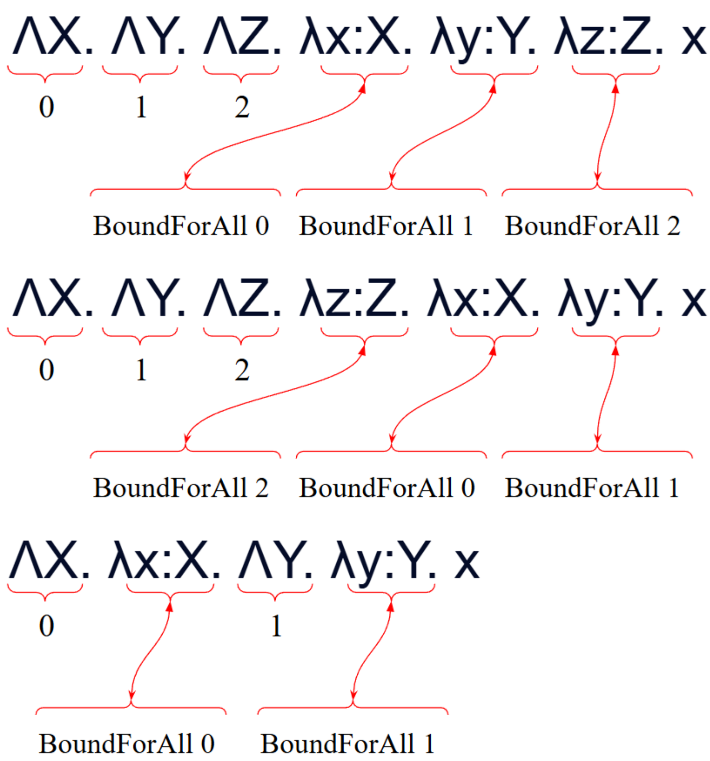
De este definición podemos ver que hay más elementos en Type que en la gramática de tipos, esto se debe a que se agregaron 2 tipos más los cuales son BoundForAll y ListEmpty.

Como se van a poder usar listas de todo tipo surge un problema a la hora de usar una lista vacía, ya que se debería especificar el tipo, a pesar de estar vacía.

Para evitar y simplificar esto lo que decidió fue darle un tipo especial a la lista vacía, **ListTEmpty**, de esta forma se evita tener que darle un tipo especifico.

Pero esto trajo problemas, por ejemplo en la función infer' en el caso de RL o en la función match en el caso de compara el tipo de una lista vacía con un tipo que es una lista pero no vacía. Para solucionar este inconveniente lo que se hizo fue separar en casos, uno es cuando la lista es vacía y el otro cuando no es una lista vacía.

En el caso del BoundForAll, este no es un tipo per se, sino que su función es similar a la idea de los indices de De Brujin. Esto seria, coloquialmente hablando, indicar a que **para todo** esta ligada la variable cuantificada, la idea es la siguiente:



En un inicio la idea era poner el BoundForAll con los Term (como el Bound que esta en Term), pero como esta idea esta relacionada con los tipos resulto más practico agregarlo aca.

Para las expresiones del lambda calculo se tiene:

```
data LamTerm  =  LVar String
              |  LAbs String Type LamTerm
              |  LApp LamTerm LamTerm
```

```
|  LTAbs String LamTerm
|  LTApp LamTerm Type
|  LTrue
|  LFalse
|  LIfThenElse LamTerm LamTerm LamTerm
|  LZero
|  LSuc LamTerm
|  LRec LamTerm LamTerm LamTerm
|  LNil
|  LCons LamTerm LamTerm
|  LRecL LamTerm LamTerm LamTerm
deriving (Show, Eq)
```

Al igual que en el Trabajo Practico 2 [1] surge el problema del uso de nombre de variables, al momento de realizar operaciones como la sustitución. Para arreglar esto se mantiene la misma idea de usar la representación con **indices de De Brujin**.

Al usar una representación sin nombre surge el problema de no tener variables libres, para evitar este inconveniente se utiliza la representación localmente sin nombres (donde variables libres y ligadas están en diferentes categorías sintácticas).

Al utilizar esta representación los términos quedan asi:

```
data Term  = Bound Int
           | Free Name
           | Term :@: Term
           | Lam Type Term
           | ForAll Term
           | TApp Term Type
           | T
           | F
           | IfThenElse Term Term Term
           | Zero
           | Suc Term
           | Rec Term Term Term
           | Nil
           | Cons Term Term
           | RecL Term Term Term
           deriving (Show, Eq)
```

Un tema que es interesante de explicar es cuando, hablando coloquialmente, el tipo de una variable tiene un para todos (es polimorfico), veamos esto con un ejemplo. Sea la siguiente expresión:

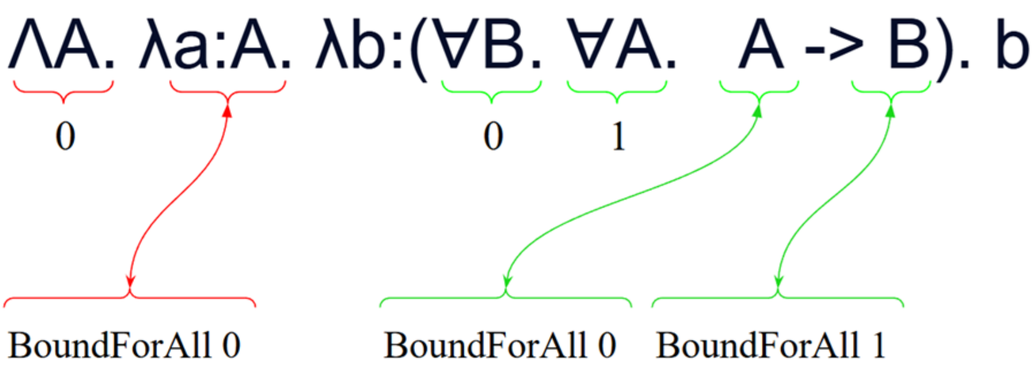
(/\A. \a:A. \b:(\/B. \/A. A -> B) . b)

Tenemos que b es una función polimorfica, al obtenga su forma en Term se va a tener lo siguiente

ForAll (Lam (BoundForAll 0) (Lam (ForAllT (ForAllT (FunT (BoundForAll 0) (BoundForAll 1)))) (Bound 0)))

Donde se puede apreciar una cosa interesantes, se tiene un solo ForAll pero tenemos (BoundForAll 0) y (BoundForAll 1). Uno tendería a pensar que esto es un error, pero no lo es ya que estos BoundForAll no están 'ligados' al ForAll sino al ForAllT.

Es decir hay una distinción entre el ForAll y ForAllT al momento de usar variables cuantificadas, uno para los Type y otro para los Term. Gráficamente seria:



4.1.1. Evaluación

Para la evaluación el interprete sigue el orden de reducción **call-by-value** en donde tenemos las siguientes reglas, las cuales son las presentes en el TP Nº2 [1] y en el material de clase del Sistema F [4]:

$$\frac{t_1 \to t_1'}{t_1\ t_2 \to t_1'\ t_2}\ \text{(E-App1)} \qquad \frac{t_2 \to t_2'}{v\ t_2 \to v\ t_2'}\ \text{(E-App2)} \qquad \frac{}{(\lambda x : T_1\ .\ t_1)v \to t_1[x/v]}\ \text{(E-AppAbs)}$$

$$\frac{ifthenelse\ T\ t_2\ t_3}{t_2}\ \text{E-IFTrue} \qquad \frac{ifthenelse\ F\ t_2\ t_3}{t_3}\ \text{E-IFFalse}$$

$$\frac{t_1 \to t_1'}{ifthenelse\ t_1\ t_2\ t_3 \to ifthenelse\ t_1'\ t_2\ t_3}\ \text{E-IF}$$

$$\frac{}{R\ t_1\ t_2\ 0 \to t_1}\ \text{E-RZero} \qquad \frac{}{R\ t_1\ t_2 (suc\ t) \to t_2 (R\ t_1\ t_2\ t) t}\ \text{E-RSuc} \qquad \frac{t_3 \to t_3'}{R\ t_1\ t_2\ t_3 \to R\ t_1\ t_2\ t_3'}\ \text{E-R}$$

$$\frac{}{RL\ t_1\ t_2\ nil \to t_1}\ \text{E-RNil} \qquad \frac{}{RL\ t_1\ t_2 (cons\ t\ l) \to t_2\ t\ l\ (RL\ t_1\ t_2\ l)}\ \text{E-RCons}$$

$$\frac{t_3 \to t_3'}{RL\ t_1\ t_2\ t_3 \to RL\ t_1\ t_2\ t_3'}\ \text{E-RL}$$

$$\frac{t_1 \to t_1'}{cons\ t_1\ t_2 \to cons\ t_1'\ t_2}\ \text{E-Cons1} \qquad \frac{t_2 \to t_2'}{cons\ t_1\ t_2 \to cons\ t_1\ t_2'}\ \text{E-Cons2}$$

$$\frac{t_1 \to t_1'}{t_1\ \langle T \rangle \to t_1'\ \langle T \rangle}\ \text{E-TApp} \qquad \frac{}{(\Lambda X\ .\ t)\ \langle T \rangle \to t[T/X]}\ \text{E-TAppAbs}$$

4.1.2. Tipos

Para realizar la inferencia de tipo usamos las siguientes reglas, las cuales al igual que en la sección anterior son las presentes en el TP Nº2 [1] y en el material de clase del Sistema F [4]:

$$\frac{}{\Gamma \vdash T : Bool}\ \text{T-True} \qquad \frac{}{\Gamma \vdash F : Bool}\ \text{T-False} \qquad \frac{\Gamma \vdash t_1 : Bool \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash ifthenelse\ t_1\ t_2\ t_3\ :\ T}\ \text{T-IF}$$

$$\frac{}{\Gamma \vdash 0 : Nat}\ \text{T-Zero} \qquad \frac{\Gamma \vdash t : Nat}{\Gamma \vdash suc\ t : Nat}\ \text{T-Suc}$$

$$\frac{\Gamma \vdash t_1 : Nat \quad \Gamma \vdash t_2 : T\ \to\ Nat \to\ T \quad \Gamma \vdash t_3 : Nat}{\Gamma \vdash R\ t_1\ t_2\ t_3 : T}\ \text{T-Rec}$$

$$\frac{}{\Gamma \vdash nil : ListEmpty}\ \text{T-Nil} \qquad \frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : List\ T}{\Gamma \vdash cons\ t_1\ t_2 : List\ T}\ \text{T-Cons}$$

$$\frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T_1 \ \rightarrow \ List\ T_1 \rightarrow \ T \rightarrow \ T \quad \Gamma \vdash t_3 : List\ T_1}{\Gamma \vdash \ RL\ t_1\ t_2\ t_3 \ : \ T}\text{T-RL}$$

$$\frac{\Gamma, X \vdash t : T}{\Gamma \vdash \Lambda X\ .\ t : \forall X\ .\ T}\text{T-TAbs} \qquad \frac{\Gamma \vdash t_1 : \forall X\ .\ T}{\Gamma \vdash t_1\langle T_2 \rangle : T[T_2/X]}\text{T-TApp}$$

4.2. Mostrar terminos

Al igual que en el TP N$^{\underline{o}}$2 [1] se va a utilizar la biblioteca **pretty printing** para mostrar por pantalla. En el archivo **src/PrettyPrinter.hs** es donde se implementa el **pretty printing** para el sistema F.

4.3. Ejmplos con resultados

Una vez que se haya compilado y ejecutado el programa nos aparecerá en la consola esto:

```
Intérprete de Sistema F
Escriba :help para recibir ayuda.
SF>
```

Luego si se quiere evaluar una expresión del Sistema F, se la ingresa por teclado, se presiona el enter y listo (también hay más opciones como el :print para mostrar los ASTs y el :type para ver el tipo de la expresión, entre muchas otras).

Una cosa a mencionar en el tema de los Bool es que si bien en la gramática el if aparece como **ifthenelse** t_1 t_2 t_3 y tanto en los LamTerm como en los Term también aparece asi, a la hora de escribirlo y mostrarlo por consola es **if** t_1 **then** t_2 **else** t_3, este cambio se hizo por comodidad, es más común escribirlo como la segunda forma que como la primera.

Veamos ejemplos (estos ejemplos están en el archivo Ejemplos.txt para que puedan ser testeados sin problemas por el lector):

4.3.1. Funcion identidad polimorfica

En el Sistema F se escribiría: ΛX. λx:X. x
En la consola escribimos: /\X. \x:X . x
Si quisiéramos evaluarla a un natural escribimos: (/\X. \x:X . x) <Nat> (suc 0)
El cual se reduce a: suc 0

4.3.2. Funcion length para listas polimorfica

En el Sistema F se escribiría: $\Lambda X.\ \lambda xs : List\ X.$ RL 0 (λx:X .ys:List X .r:Nat .$suc\ r$) xs
En la consola escribimos: (/\X. (\xs:List X. RL 0 (\x:X .\ys:List X. \r:Nat. suc r) xs))
Si quisiéramos evaluarla a una lista de funciones polimorficas escribimos: ((/\X. (\xs:List X. RL 0 (\x:X .\ys:List X. \r:Nat. suc r) xs)) <\/X. X -> X>) (cons (/\X. \x:X. x) cons (/\X. \x:X. x) nil)
El cual se reduce a: suc suc 0

(Si se prueba con nil da como resultado 0)

4.3.3. Funcion que toma como argumento una funcion polimorfica

En el Sistema F se escribiría: $\Lambda A.\ \lambda a : A.\ \lambda b : (\forall B.B \to B).\ b$
En la consola escribimos: (/\A. \a:A. \b:(\/B. B -> B) . b)
Si quisiéramos evaluarla podría ser algo asi: (((((/\A. \a:A. \b:(\/B. B -> B) . b) <Nat>) 0) (/ \X. \x:X. x)) <Bool>) T
El cual se reduce a: T

Referencias

[1] Cátedra de Análisis del Lenguaje de Programación. Trabajo practico nº2. *Departamento de Ciencias de la Computación*, 2024.

[2] Mike Pilgrem. Stack documentation, haskell. `https://docs.haskellstack.org/en/stable/`.

[3] Andy Gill and Simon Marlow. Happy documentation, haskell. `https://haskell-happy.readthedocs.io/en/latest/`.

[4] Cátedra de Análisis del Lenguaje de Programación. Polimorfismo. *Departamento de Ciencias de la Computación*, 2024.